Towards Efficient Neural Machine Translation

Xiang Kong

CMU-LTI-22-010

May 2022

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Eduard Hovy (Chair)	Carnegie Mellon University
Emma Strubell	Carnegie Mellon University
Alon Lavie	Carnegie Mellon University
Alexander Rush	Cornell University

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

Keywords: Neural Machine Translation, Efficiency, Multilingual Neural Machine Translation, Non-autoregressive Translation

Acknowledgments

I would like to express my deepest gratitude to my advisor Eduard Hovy (Ed). I am grateful to meet him during my PhD journey. Ed is an expert in natural language processing (NLP) and more importantly, he is a superb advisor. It is he who guides me to study the essential aspects of NLP research from our first meeting. I can always learn so many new things and insightful ideas which motivate me to pay attention to research that goes beyond just having higher numbers. Furthermore, he encourages me to explore various topics which I am interested in. Therefore, during my PhD study, I explored many different directions and tasks under Ed's guidance, in which I built the skillsets to quickly adapt to a new domain. Besides my academic achievements, he also cares about my long-term successes and gives me invaluable suggestions to my career and life. I am very grateful to have Ed as my advisor in my PhD journey.

I would like to thank my committee members, Emma Strubell, Alon Lavie and Alexander Rush, for providing enlightening feedback and suggestions. Their comments have made this thesis much more complete and led to many potential research projects. In particular, Emma Strubell helps me to refine the definition of efficiency in this thesis, making the contribution clear. Alon Lavie is an expert in machine translation evaluation who helps to compare proposed NMT systems thoroughly. Alexander Rush is an expert on structure prediction, text generation, etc, who helps me to make the model description accurate.

I would also like to thank people in Edvisees group. It is such an open environment where we can discuss many research projects from several perspectives and life. These discussions inspire me to conduct a lot of interesting research. We often go for dinner together. Thanks to past and present members of Edvisees group for making this such a great environment: Zhengzhong Liu, Xuezhe Ma, Qizhe Xie, Pradeep Dasigi, Diyi Yang, Nicolas Fauceglia, Yiu-Chang Lin, Jun Araki, Dongyeop Kang, Varun Gangal, Jiarui Xu, Xianyang Chen, Zhisong Zhang and Steven Feng.

I also want to thank the staff at LTI especially Kate Schaich and Stacey Young who endeavour to students feel at home. I always get prompt and useful answers from them to my questions regrading course registration, immigration, etc. They are always friendly to all students and deeply care about students' well-being.

Collaboration is a big lesson that I learned, and also a fun part of graduate school. I would also like to express my gratitude to many amazing people at CMU who I had the pleasure to be friend with, work with or learn interesting ideas from: Qizhe Xie, Xuezhe Ma, Jiatao Gu, Zhisong Zhang, Varun Gangal, Xianyang Chen, Zhengzhong Liu, Liangke Gui, Guokun Lai, JingZhou Liu, Xin Qian, Yuexin Wu, Bohan Li, Yuqing Tang, Chau Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, Angela Fan, Mengzhou Xia, Xiang Kong, Antonios Anastasopoulos, Graham Neubig, Shanghang Zhang, Zhaopeng Tu, Shuming Shi, Tong Zhang, Asa Cooper Stickland, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, Luke Zettlemoyer, Xuesong Yang, Mark Hasegawa-Johnson, Jeung-Yoon Choi, Stefanie Shattuck-Hufnagel, Yanlu Xie, Jaime G Carbonell, Hans Chalupsky, Anatole Gershman, Alex Hauptmann, Florian Metze, Teruko Mitamura, Zaid Sheikh, Ankit Dangi, Aditi Chaudhary, Xianyang Chen, Bernie Huang, Salvador Medina, Hector Liu, Xuezhe Ma, Maria Ryskina, Ramon Sanabria.

I have the deepest appreciation for my parents and my grandparents for unconditional love, guidance and freedom they gave me. I dedicate this thesis to you, in love and appreciation.

Finally, I want to thank my dear wife, Jialu Zhao. Thank you for sharing my happiness and difficulties along the way. Thank you for your unconditional love and support all time. I couldn’t have done this thesis without you. Let us start a new chapter of our life to explore the beautiful in this world together.

Abstract

Machine translation (MT), the use of machines to automatically translate from one language to others, aims to overcome language barriers among people from different cultures. Recently, neural networks-based machine translation (NMT) models significantly narrow the gap between machine and human translations in terms of translation accuracy. However, some new challenges [90] are also introduced and **efficiency** is one of the most important issues. Specifically, with a complex deep network structure, NMT models generally have high space and computational costs, hindering their deployment in real-time applications with strict latency requirements or devices with limited memory resources. In this thesis, we aim to improve the decoding efficiency of NMT from three aspects, (1) **computational efficiency**: NMT, similar to other deep learning models, employs a deep network structure with a large number of parameters and high model complexity, resulting in high memory usage and a relatively slow decoding process; (2) **decoding parallelizability efficiency**: another main reason of the low-speed inference process for NMT is the autoregressive property of its decoder that only generates one token at a time and can not be parallelizable; (3) **efficiency in multilingual NMT**: to better support translations between multiple languages, a popular strategy is to employ a deeper encoder and decoder structure with the increased model capacity to handle multiple languages. However, the extra latency and memory costs introduced by this approach make it unacceptable for efficiency-constrained applications.

This thesis consists of three parts to tackle these challenges respectively. First, to improve the computational efficiency, we focus on some modules of NMT and develop novel structures and learning algorithms including (1) investigating word encoding mechanisms to significantly reduce the time and space consumption of the embedding and softmax layers; (2) developing a linear unified nested attention mechanism that approximates regular attention, yielding only linear (as opposed to quadratic) time and space complexity. Then we relieve the autoregressive property in the conventional NMT decoding algorithm to speed up the decoding process, which includes (1) designing a semi-autoregressive decoding algorithm which keeps the autoregressive property locally but avoids it globally; (2) developing a fully non-autoregressive translation system in which all tokens are generated in parallel. Finally, we investigate the decoding efficiency in the multilingual translation scenario which consists of (1) studying the speed-accuracy trade-off for multilingual translation; (2) improving the decoding speed through the model capacity allocations from different granularity while maintaining superior translation quality.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Building automatic translation systems to overcome language barriers and build bridges between people from different cultures has a long history. A robust machine translation (MT) engine which is capable of accurately and **rapidly** translating one language to another can benefit people across the globe. Since 1950s, scientists have been devoted to proposing various practical systems such as rule-based MT [42, 176], example-based MT [126, 162] and statistical MT [13, 89, 91].

In recent years, as a new paradigm, neural networks-based machine translation (NMT) models [7, 22, 168, 178] have achieved state-of-the-art translation performance and significantly narrowed the gap between machine and human translations in terms of translation quality between some languages [59, 190]. Several open-source NMT frameworks [69, 87, 132, 153] also facilitate the research and development in this field. In contrast to the statistical machine translation model which consists of various separate designed sub-components, NMT employs a single neural network to directly learn the mapping between source sentences and their translations through an end-to-end training process. Concretely, the goal of NMT models is learning a conditional language model, i.e., the probability of the target sentence given the source sentence. The great success of NMT is mainly due to its strong ability to auto-decompose and represent millions of translation combinations. To achieve this, NMT tends to employ a deep neural network structure with high complexity computations and a massive number of parameters. For example, a standard transformer-base [178] model with a vocabulary of 40,000 tokens contains around 64 million parameters.

Despite the impressive progress in translation quality brought by the power of NMT, new challenges are also introduced [90]. An important one among them is **efficiency** which includes several aspects such as data efficiency, training efficiency and inference efficiency. Specifically, in this thesis, we mainly focus on the *inference* efficiency from several scenarios and aspects.

1.1 Research Objective

The complicated structure and the large capacity of deep neural networks are the main reasons for NMT to surpass other machine translation models. Nonetheless, this architecture also renders the NMT model computation-expensive and memory-intensive. As a result, during the inference process, compared to the conventional MT models such as phrased-based statistical machine translation, generating sentences by NMT is much slower. Also, the large number of parameters and high computational complexity bring high memory usage. This low inference efficiency profoundly affects the smoothness of the communication and the real-time application with strict latency or memory requirements. Therefore, in this thesis, we specifically investigate methods to ameliorate the decoding speed or memory consumption of NMT from three aspects:

Computational Complexity: The NMT model consists of several modules with various operations. For instance, the softmax function at the final layer of NMT is employed to compute scores for each candidate word and normalize them to probabilities. Its computation involves a matrix multiplication proportional to the size of the output vocabulary V. Since NMT models typically have tens of thousands of words, this module becomes a bottleneck in terms of speed and memory. Another module we investigate in this thesis is the attention mechanism, the core function in transformer [178], which is to model pairwise interactions between tokens. While attention is powerful, its quadratic time and space complexity w.r.t. the length of the input sequence, prohibitively restricts their potential application to tasks requiring longer input sequences.

Decoding Parallelizability: Besides ameliorating the computational complexity of some modules in NMT, another aspect to speed up the decoding process is to mitigate the low parallelizability of its standard decoding algorithm. At inference time, standard autoregressive translation (AT) models only generate one token at a time conditioning on the target sequence produced so far and the whole source sentence. This decoding process can not be parallelizable, and, in the case of NMT models, it becomes slow because a computationally intensive neural network is used to generate each token. Therefore, this autoregressive decoding process is still the main bottleneck of inference speed and it is meaningful to make this process as parallelizable as possible.

Capacity Allocation in Multilingual Translation Besides improving the parallelism of the NMT decoding process, another method is to exploit the parallelization difference between transformer encoder and decoder at inference time. Specifically, due to the autoregressive property, the decoder needs to generate tokens one by one. However, the computation in the encoder is still parallelized over the source sentence. Therefore, the main latency of the transformer at inference time happens in the decoder, especially translating long sentences. Recently, Kasai et al. [73] find that on bilingual machine translation tasks, putting more capacity of the transformer model to the encoder from the decoder substantially reduces the decoding time and maintains performance at the same time. Therefore, we aim to understand the effect of layer allocations on the multilingual neural machine translation task.

Overall, in this thesis, we seek to tackle the aforementioned challenges to improve the latency and memory usage of the NMT model while maintaining its superior translation quality at the same time.

1.2 Thesis Outline

In Chapter 2, we briefly review the background of this thesis including the architecture, training and inference of neural machine translation models.

Following the inference efficiency-related challenges discussed above, this thesis is composed of three main parts.

In Part I, we endeavor to improve the computational efficiency issue by optimizing various modules in NMT.

In Chapter 3, we focus on the vocabulary representation. We observe that due to the large vocabulary size in language generation models, the embedding layer occupies the majority of the model parameters. Besides, it makes the softmax layer computation-expensive. To reduce the number of parameters for word representations in NMT models, we investigate two improved word coding methods, i.e., LightRNN [108] and Byte Pair Encoding (BPE) [154], which effectively relieves the memory and computation burden. Furthermore, BPE achieves the best performance due to its strong ability to handle rare and out-of-vocabulary words. In Chapter 4, we aim to reduce the time and space complexity of the standard attention mechanism in transformer [178] and introduce Luna which uses two nested attention functions to approximate the standard attention. Specifically, with the first attention function, Luna packs the input sequence into a packed sequence. Then, the packed sequence is unpacked using the second attention function. Importantly, we model this packed sequence as an extra input sequence which can adequately encode the contextual information of the real input sequence.

In Part II, we work on improving the low parallelizability of the standard autoregressive decoding algorithm in which sentences are generated token by token.

In Chapter 5, we introduce a semi-autoregressive neural machine translation model with a novel local autoregressive translation (LAT) mechanism which captures local dependencies among target outputs. With LAT, we first generate a bunch of short sequences in parallel and each sequence is generated autoregressively, then an efficient merging algorithm is designed to align and merge the output pieces into the final translation. In Chapter 6, we aim to develop a competitive fully non-autoregressive neural machine translation (NAT) system which generates tokens in parallel. We find that the key to successfully training a fully NAT model is to reduce the target dependency as much as possible. Therefore, several target dependency reduction techniques are applied to minimize the performance gap between fully NAT and AT models while significantly improving the decoding speed.

In Part III, we strive to improve the inference efficiency in the multilingual neural machine translation (MNMT) scenario through capacity allocations of the transformer.

In Chapter 7, we extensively study the speed-accuracy trade-off of MNMT models through model capacity allocations. We further design a deep encoder, multiple shallow decoders (DEMSD) model to accelerate the decoding speed compared to a standard MNMT model with competitive translation quality on the one-to-many translation task. In Chapter 8, inference-efficient mixture-of-experts (MoEs) are applied to boost the performance of the shallow decoder-based models on one-to-many translation tasks. Moreover, we examine the capacity bottleneck of the shallow decoder on the one-to-many translation task through an empirical study.

Chapter 2

Background

In this chapter, we briefly review some aspects of the standard neural machine translation (NMT) model, including its modeling, training and inference. This chapter also briefly surveys previous works related to inference-efficient neural machine translation.

2.1 Modeling

Neural Language Modeling

Prior to introducing the neural machine translation system, as the basis of many natural language tasks, we first review the language modeling using various kinds of neural networks, i.e., neural network language modeling (NNLM). For instance, Bengio et al. [11] introduce the first NNLM which employs feed-forward neural networks to predict tokens given words in a fixed-size window. Formally, given a vocabulary of all possible words, V, for a sentence $s = (w_1, ..., w_i, ..., w_n)$, NNLM can assign the probability $p(s;\theta) = p(w_1, ..., w_i, ..., w_n;\theta)$ to it with parameters θ. However, it is difficult to model the probability of the entire sentence directly so approximate methods are necessary. The most popular and widely used approximation method is to model NNLM in an autoregressive way, where the probability of the current word conditions on its previous words. Therefore, the probability of the given sentence is reformulated to: $p(w_1, ..., w_i, ..., w_n;\theta) = \prod_{i=1}^{n} p(w_i|w_1 : w_{i-1};\theta)$.

Conditional Neural Language Modeling

With neural language modeling, sentences can be generated from scratch and given a sentence, we can also know how likely it is in that language. In practice, there is also a need for generating meaningful sentences conditioning on the context. For example, machine translation models need to take the source sentence into account. In dialogue generation, the generated response needs to be coherent with its prompts. Therefore, the conditional autoregressive neural language modeling is introduced through the sequence to sequence (seq2seq) learning schema [169] where the input sequence is mapped to a fixed-size vector and each token in the target sequence will be decoded from this vector. Formally, given the input sequence x and the target sequence y, the probability of y given x from a model with parameters θ is: $p(y|x) = \prod_{i=1}^{n} p(y_i|y_{<i}; x; \theta)$.

Neural Machine Translation and Attention Mechanism

Almost all neural machine translation models are built based on the sequence to sequence learning schema through various kinds of networks, e.g., recurrent neural networks (RNNs) [23, 169], convolutional neural networks (CNNs) [45] and the transformer model [178]. All of them can be regarded as conditional neural language modeling. Specifically, the input sentence from a source language is converted to a fixed-size vector through a neural network, a.k.a. encoder, then another network called decoder will generate output in an autoregressive way conditioning on the partial translation and the context vector.

One main bottleneck of this vanilla seq2seq model is the need for the encoder network to be able to compress all information in the source sentence into a fixed-size context vector especially for long sentences. Inspired by the alignment between words in source and target sequences in statistical machine translation [88], [7] propose the attention mechanism to explicitly (soft-)align input and output sequences. Rather than predicting target words by a single fixed vector, the attention mechanism outputs specific context vectors dynamically for each output time step. Intuitively, with the attention mechanism, the model concentrates on different parts of the source sentences when generating target tokens at various time steps. Furthermore, the attention mechanism is a valuable breakthrough in deep learning research, which not only helps neural machine translation models to achieve state-of-the-art performance, but spawns the rise of many important models recently such as the transformer [178] and BERT [33] as well.

Multilingual Neural Machine Translation (MNMT)

After obtaining impressive performance on bilingual translation tasks [7, 23, 169], neural machine translation has been naturally extended to the multilingual machine translation scenario. To support many language pairs within a single model, [68] prepend a token to the source sentence to specify the target language.

There are several advantages to developing one NMT model for translation between many languages. For example, training models with data from multiple language pairs enable the knowledge transfer between them which can boost the performance of low-resource languages.

Another one is that the MNMT model is capable of translating between unseen language pairs during training, a.k.a. zero-shot translation.

Despite these potential benefits, MNMT generally performs worse than its corresponding bilingual NMT models and this performance gap will become larger as the number of language pairs increases [1, 4, 68]. Suspecting that poor performance is due to the limited model capacity, many prior works adopt deeper encoder and decoder to overcome this capacity bottleneck [182, 198, 199]. Nonetheless, the increased model capacity leads to a high latency decoding process with large memory consumption.

2.2 Training

Vocabulary Encoding

Typically, neural machine translation operates with a fixed word-level vocabulary and a special symbol, <UNK>, is introduced to represent unseen tokens. However, machine translation is an open-vocabulary problem and it is necessary to make the NMT model capable of open-vocabulary translation. One popular method to handle this open-vocabulary issue is to learn subword units by Byte Pair Encoding (BPE) [44, 95, 154]. Specifically, each word in the training corpus will be split into several subword components and NMT models will be operated on this subword-level vocabulary. Furthermore, there are also some works [79, 101] directly learn NMT models on a character-level vocabulary.

Objective Function

Neural Machine Translation (NMT) is an end-to-end structure which could directly model the translation probability between a source sentence $x = x_1, x_2, \dots, x_J$ and a target sentence $y = y_1, y_2, \dots, y_I$ token by token:

$$p(y|x) = \prod_{i=1}^{I} p(y_i|y_{<i}, x; \theta) \tag{2.1}$$

where $y_{<i} = \{y_1, y_2, \dots, y_{i-1}\}$ is the partial translation before decoding step i and θ represents NMT's parameters. The probability of generating the i-th word $p(y_i|y_{<i}, x; \theta)$ is calculated by

$$p(y_i|y_{<i}, x; \theta) \propto \exp\{f(y_{i-1}, s_i, c_i; \theta)\} \tag{2.2}$$

where s_i is the i-th hidden state of the decoder and $f(\cdot)$ is a non-linear activation function of the decoder state. c_i is a distinct source representation for time i, calculated as a weighted sum of the source annotations: $c_i = \sum_{j=1}^{J} \alpha_{i,j} \cdot h_j$, where h_j is the annotation of x_j from an encoder, and its weight $\alpha_{i,j}$ is computed by

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{j'=1}^{J} \exp(e_{i,j'})} \quad with \;\; e_{i,j} = a(s_{i-1}, h_j) \tag{2.3}$$

where $a(\cdot)$ is an *attention model* that scores how well y_i and h_j (*i.e.,* x_j) match, and $\alpha_{i,j}$ is the normalized alignment probability of x_j being aligned to y_i.

Finally, the parameters of NMT θ are trained to maximize the likelihood of training instances $\{[x^n, y^n]\}_{n=1}^{N}$:

$$L(\theta) = \arg\max_{\theta} \sum_{n=1}^{N} \log P(y^n | x^n; \theta) \tag{2.4}$$

2.3 Inference

It is vital to design inference algorithms to accurately and rapidly generate high-quality translations given input sequences. In general, NMT models attempt to produce the most likely translation given an input sequence, i.e.,

$$\hat{y} = \arg\max_{y} p(y|x; \theta) = \arg\max_{y_1,\ldots,y_I} \prod_{i=1}^{I} p(y_i | y_{<i}; x; \theta) \tag{2.5}$$

The most likely translation given the current model could be obtained by computing all possible sequences, which, however, is computationally intractable as the number of possible candidates grows exponentially with respect to the sequence length. Therefore, some approximate decoding algorithms such as greedy and beam search methods are applied in practice.

Greedy Inference

In the greedy decoding method, at each time step, we simply choose the token with the highest conditional probability as the final translation and use it to predict the following tokens. The translated word at each time step t will be formulated as:

$$\hat{y}_t = \arg\max_{y} p(y_t | \hat{y}_{<t}; x; \theta) \tag{2.6}$$

The greedy decoding process is fast and straightforward but it is generally error-propagated and sub-optimal.

Beam Search Inference

To balance the translation accuracy and computational complexity, the beam search decoding method is designed to generate high-quality sentences with tractable computation. During the decoding process, k possible candidates are maintained. At each time step, all successors of k states are generated and the k best successors are selected for the following inference.

Thanks to the larger searching space, the beam search inference method generally performs better than the greedy decoding algorithm. However, the computational complexity grows linearly with the size of beam and [90] show that beam search decoding only improves translation quality for narrow beams and deteriorates when exposed to the larger search space.

2.4 Related Work

Breaking the Autoregressive Property of the Decoding Process

One main advantage of transformer [178] over recurrent neural networks is its higher computation parallelism during training time. However, the standard transformer-based NMT model generates sequences autoregressively at inference time. Therefore, one line of research towards inference-efficient NMT is to break the autoregressive property and enable the decoding processing as parallel as possible.

Non-autoregressive translation models aim to generate sequences in parallel. Gu et al. [55] introduce a set of latent variables to model the fertilities of source words to tackle the multimodality problem in non-autoregressive translation and they find that it is effective to train NAT models on distilled data using sequence-level knowledge distillation [78]. Several works find that the standard cross entropy loss is too strict for NAT models. In order to make the loss function more compatible with NAT models, several works [49, 110, 150] use the CTC loss or aligned cross entropy to introduce latent alignments between generated and golden sequences. Shao et al. [156] use the bag-of-2grams loss to mitigate the overcorrection effect of the cross entropy loss. Wang et al. [185] incorporate two auxiliary regularization terms in the training process of NAT models to mitigate the incomplete and repeated translation issues. Sun et al. [167] aim to directly model the multimodal issue in the target token spaces of NAT and design an efficient approximation for CRF and a dynamic transition method to handle positional contexts. Several works find that giving partial golden translations to the NAT decoder can help the NAT training process and Qian et al. [138] optimize the number of glancing targets to NAT models according to the quality of generated sentences to reduce the mismatch between training and inference.

Another popular solution to improve the translation accuracy of NAT models is to sacrifice the speed-up by incorporating an iterative refinement process. Lee et al. [102] propose a non-autoregressive sequence model based on an iterative refinement process. The model can be viewed as both a latent variable model and a conditional denoising autoencoder and it is optimized by the hybrid of lower bound maximization and reconstruction error minimization. Stern et al. [163] introduce a blockwise parallel decoding scheme in which predictions for multiple time steps are made in parallel then back off to the longest prefix validated by a scoring model. Kaiser et al. [70] use latent variables to make the decoding process more parallel. They encode the target sentence into a shorter sequence of discrete latent variables. The final target sequence will be decoded from this latent sequence in parallel. Note that during inference time, this sequence of latent variables is generated autoregressively. Insertion Transformer [165] accommodates arbitrary orderings by allowing for tokens to be inserted anywhere in the sequence during decoding instead of relying on a fixed, left-to-right ordering of outputs. During inference, output tokens could be generated in a sequential manner for autoregressive decoding or in a semi-autoregressive style with parallel decoding through inserting multiple positions simultaneously. Similarly, Chan et al. [15] also introduce a simple insertion-based approach to generative modeling for sequences and sequence pairs. Inspired by BERT [33], Ghazvininejad et al. [46] propose a conditional masked language modelling and design an interactive mask predict decoding algorithm. In one translation iteration, all tokens are generated in parallel and tokens with lower confidence scores will be masked again for re-prediction. Ma et al. [117] build an iterative NAT with latent variables, the conditional density of which is modelled by normalizing flows. Tu et al. [177] train a non-autoregressive machine translation model to minimize the energy defined by a pretrained autoregressive model. Gu et al. [56] design a sequence generation model consisting of insertion and deletion operations. Tokens are generated in parallel in each of these stages. Shu et al. [160] propose a continuous latent variables-based NAT model to improve the performance with a deterministic inference algorithm. Latent variables are iteratively refined to generate higher-quality sequences. Saharia et al. [150] adopt Imputer [16] to translation problems and demonstrate the effectiveness of latent alignments in Imputer with iterative decoding methods. Noting that conditional random fields with bounded context can be decoded in parallel, Deng and Rush [30] introduce the Markov transformer and an efficient cascaded decoding approach.

Optimizing autoregressive NMT

Prior work has suggested various ways to optimize autoregressive transformer-based NMT for fast inference and reduced memory consumption.

Due to the quadratic time complexity of the self-attention mechanism, several works propose various methods to reduce its complexity. Zhang et al. [197] design an average attention network to speed up the attention computation. Wu et al. [189] argue that a lightweight conv layer which has linear time complexity is able to achieve comparable results with self-attention. Xiao et al. [193] propose a fast and lightweight attention model which shares attention weights between adjacent layers and enables the efficient re-use of hidden states in a vertical manner. Raganato et al. [142], You et al. [196] find that attention heads in the multi-head attention mechanism mostly focus on local windows so they employ non-learned, fixed attention weights to replace the traditional learned attention heads. Competitive translation accuracy especially on low-resource languages is obtained via hard-coded attention heads with just one learned head. Peng et al. [135] use random feature methods to approximate the softmax function, and apply them to machine translation tasks to improve the decoding speed.

Shi and Knight [159] accelerate the decoding process through reducing the vocabulary size by word alignments. Wang et al. [181] replace the transformer decoder with a LSTM-based decoder while keeping the transformer encoder. They find that this hybrid architecture is able to achieve competitive results with a 2 times speed-up. Kasai et al. [73] employ a shallow decoder to speed up the inference processing on CPUs or GPUs. They also find that shallow decoder NAT significantly degrades accuracy. Deng and Rush [31] replace the decoder in NAT model with a search lattice. They first construct a candidate lattice using efficient lookup operations, generate lattice scores from a deep encoder, and finally find the best path using dynamic programming.

Several general techniques such as knowledge distillation [64, 78], model pruning, quantization, etc, are also applied to reduce the model size and memory consumption. See et al. [152] investigate magnitude-based pruning schemes to compress NMT models, namely class-blind, class-uniform, and class-distribution, which differ in terms of how pruning thresholds are computed for the different classes of weights in the NMT architecture. Michel et al. [120] find that a large proportion of attention heads can be removed at test time without significantly impacting performance, and at some layers the number of attention heads can even be reduced to a single head. Fan et al. [38] design a structure dropout, i.e., LayerDrop, which has a regularization effect during training and allows for efficient pruning at inference time. Kim et al. [80] apply several techniques to improve the decoding speed, such as pre-packed 8-bit matrix products, improved batched decoding, cache-friendly student architectures with parameter sharing and light-weight RNN-based decoder architectures. Junczys-Dowmunt et al. [69] build an efficient NMT written in pure C++ with minimal dependencies. A lot of works apply knowledge distillation to transfer knowledge from large teacher models to compact, efficient student models [60].

Part I

Computation-efficient Neural Machine Translation

Chapter 3

Fast and Simple Mixture of Softmaxes with BPE and Hybrid-LightRNN for Language Generation

3.1 Introduction

The Sequence-to-Sequence model (seq2seq) [6, 168, 178] has led to significant research progress on conditional language generation such as neural machine translation over the last few years. Traditionally, seq2seq models apply one-hot representations (whose dimension is equal to the size of the vocabulary V) for each token in V and map this one-hot vector to a continuous vector through an embedding layer. At the prediction layer, the hidden state is projected by an output matrix followed by a softmax function to output a probability distribution over all words in V. Therefore, the computational complexity of the softmax function is proportional to the vocabulary size. With a large vocabulary which is typically the case for building natural language generation systems, the softmax layer becomes a computational bottleneck since it needs to access every token to obtain the normalization factor and finally compute the probability distribution over all words in the vocabulary. Besides the high computational complexity in the softmax layer, a large vocabulary also causes a memory issue. For instance, in a NMT model with a vocabulary of 50k, the size of embedding layers accounts for around **50%** of the total number of model parameters in a transformer-base [178] model.

To address the aforementioned drawbacks, a natural idea is to apply efficient vocabulary construction methods. On a high level, we aim at an encoding mechanism of the vocabulary so that each word can be represented as a code sequence and distinct words may share some of these codes. With this sharing mechanism, we will have a smaller candidate set, resulting in fewer model parameters and reduced time and memory consumption for the softmax layer.

Therefore, in this work, we investigate two word encoding algorithms for these purposes: the first one is called LightRNN, which *learns* an encoding mechanism from the data based on the language modeling objective. The other one is Byte Pair Encoding (BPE) [44, 155], which is originally proposed to help with translating rare words. When evaluated on machine translation (MT) and image captioning, both of these approaches can effectively reduce the time and memory consumption with similar or better performance. Furthermore, comparing these two methods, BPE-based models have significantly better performance because of its strong ability to handle rare and unknown words. Moreover, we apply mixture of softmax (MoS) [194] on these vocabulary representation methods to further improve the performance.

Our contribution is two-fold. Firstly, we propose to use Hybrid-LightRNN and BPE to create time- and memory-efficient vocabulary representations for two language generation tasks. Secondly, we demonstrate the empirical effectiveness of MoS on sentence generation by improved results on machine translation and image captioning.

3.2 Background: Mixture of Softmaxes

Mixture of Softmaxes (MoS) [194] is introduced to address the expressiveness limitations of softmax-based models. In this section, we briefly review the motivation and the formulation of MoS.

With the autoregressive factorization, a generation model estimates the distribution of the next token x given the context c. In language modeling, the context is composed of previous words of x. In conditional generation tasks such as MT or image captioning, the context also contains the source sentence or the image. Let $P^*(X \mid c_i)$ denote the ground-truth distribution of the next token given context c. Then the standard softmax function computes the probability distribution $P_\theta(x \mid c)$ as

$$P_\theta(x \mid c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}}$$

where $\mathbf{h}_c$ is the context vector and $\mathbf{w}_x$ is the word embedding.

Softmax Bottleneck Yang et al. [194] show the expressiveness limitation of the softmax function from a matrix factorization perspective. Specifically, suppose that the number of valid contexts is finite. We list all contexts as $c_1, c_2, \cdots, c_N$. Let $\mathbf{A} \in \mathbb{R}^{N \times V}, \mathbf{W} \in \mathbb{R}^{V \times d}, \mathbf{H} \in \mathbb{R}^{N \times d}$ denote the log probability of the ground-truth distribution, the word embedding matrix and the context representation matrix respectively, where N is the number of contexts, V is the vocabulary size and d is the dimensionality of the embedding vector and the context vector. In other words, $\mathbf{A}_{i,j} = \log P^*(x_j \mid c_i), \mathbf{W}_j = \mathbf{w}_{x_j}, \mathbf{H}_i = \mathbf{h}_{c_i}$.

Let $F(\mathbf{A})$ denote all matrices obtained by applying row-wise shifting to $\mathbf{A}$. Since all matrices in $F(\mathbf{A})$ result in the same probability distribution due to the normalization term in the softmax, the softmax function can output the ground-truth distribution P^* if and only if the factorization $\mathbf{H}\mathbf{W}^\top$ approximate any matrix in $F(\mathbf{A})$.

However, in language generation tasks, matrices in $F(\mathbf{A})$ cannot be approximated by $\mathbf{H}\mathbf{W}^\top$ because of the differences in their matrix ranks. More specifically, the rank of $\mathbf{H}\mathbf{W}^\top$ is limited by the embedding vector dimensionality d. In comparison, as shown in Yang et al. [194], $\mathbf{A}$ and any other matrices within $F(\mathbf{A})$ have similar high ranks since different contexts result in highly different probability distributions of the next token. Consequently, the ground-truth distribution P^* cannot be approximated by the softmax distribution P_θ, which results in the softmax Bottleneck.

MoS To tackle the softmax bottleneck problem, MoS formulate the distribution as the weighted average of K softmax components:

$$P_\theta(x \mid c) = \sum_{k=1}^{K} \pi_{c,k} \frac{\exp \mathbf{h}_{c,k}^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_{c,k}^\top \mathbf{w}_{x'}} \tag{3.1}$$

where $\pi_{c,k}$ is the mixture weight of the k-th softmax component and $\mathbf{h}_{c,k}$ is the k-th context vector. On language modeling, it has been shown empirically that such a formulation leads to a high rank matrix. Note that since all softmaxes share the same word embedding matrix, the number of parameters do not increase rapidly with more mixtures, preventing overfitting.

The mixture weight and the context vectors are computed as

$$\begin{aligned} \pi_{c,k} &= \frac{\exp \mathbf{g}^\top \mathbf{w}_k^{(\pi)}}{\sum_{k'=1}^{K} \exp \mathbf{g}^\top \mathbf{w}_{k'}^{(\pi)}} \\ \mathbf{h}_{c,k} &= \tanh(\mathbf{W}_k^{(h)} \mathbf{g}) \end{aligned} \tag{3.2}$$

where g denotes a vector representation of the context c. $\mathbf{w}^{(\pi)}$ and $\mathbf{W}^{(h)}$ denote the parameters of the mixture weight and the parameters of the context vector with a slight abuse of notation.

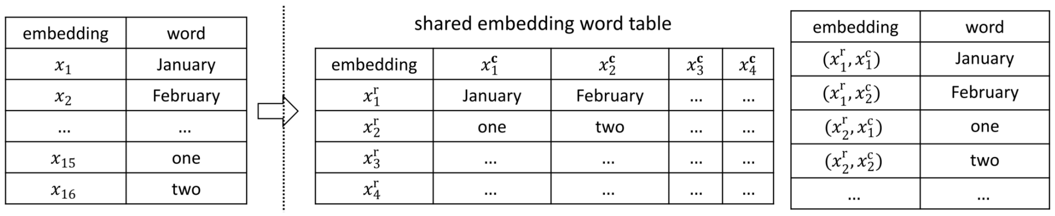
Time and Memory Cost As shown in Equation. 3.1, MoS computes K softmaxes and output the weighted average of the K probability distributions. Though MoS effectively increases the expressiveness of a generation model, it also incurs a large time and memory cost since it needs to perform K softmax operations on the whole vocabulary. The time and memory costs not only hinder rapid algorithm developments but also limit the mixture number when resources are limited, restricting the power of MoS.

3.3 Efficient Word Encoding

In this section, we introduce two word encoding algorithms to reduce the memory and time consumption of the softmax layer and the model parameter size. We aim to obtain a word encoding mechanism where the number of potential codes is much smaller than the number of word types. Therefore, in this chapter, we investigate two methods to obtain this goal which are LightRNN [108] and Byte Pair Encoding (BPE) [44, 155].

LightRNN

Li et al. [108] propose to use 2-Component (2C) shared embedding for word representations. Specifically, all words in the vocabulary are allocated into a 2d table and words in the same row or column will share a row or column embedding vector respectively. Each word in this table can be represented by its corresponding row and column vectors. An example of this table is shown in Figure 3.1. For a square word table, the length of its side is $\sqrt{|V|}$ where $|V|$ is the vocabulary



embedding	word
x_1	January
x_2	February
...	...
x_{15}	one
x_{16}	two

shared embedding word table

embedding	x_1^c	x_2^c	x_3^c	x_4^c
x_1^r	January	February	...	...
x_2^r	one	two	...	...
x_3^r	...	...	...	...
x_4^r	...	...	...	...

embedding	word
(x_1^r, x_1^c)	January
(x_1^r, x_2^c)	February
(x_2^r, x_1^c)	one
(x_2^r, x_2^c)	two
...	...

Figure 3.1: An sample of the LightRNN word table [108].

size. Therefore, only $2\sqrt{|V|}$ vectors in total are needed to represent all words in a vocabulary with $|V|$ words, and thus greatly reduce the parameter size to represent V as compared to the standard approach ($\mathcal{O}(\sqrt{|V|} * d)$ vs. $\mathcal{O}(|V| * d)$) where d is the embedding dimension. Similarly, LightRNN is also capable of circumventing the linear dependency on the vocabulary size for the computational time and memory of the softmax layer.

Based on the above idea, the key part is how to appropriately generate this table, i.e., allocating words into columns and rows. Specifically, Li et al. [108] introduce a bootstrap algorithm to iteratively refine locations of words in the table.

1. Initialize the word table by randomly putting words into the table.
2. Train a language model based on the current table allocation until convergence or reaching an exit criterion.
3. Fix the column and row vectors obtained from the above step and refine the allocation in the table through the minimum cost maximum flow (MCMF) algorithm [2].

After several rounds, Li et al. [108] show that LightRNN is able to group semantic or syntactical similar words together and we can encode the original word in the vocabulary based on the final word table.

Hybrid-LightRNN Although the LightRNN algorithm can significantly reduce the memory and time consumption with large vocabularies, if we only use $O(\sqrt{|V|})$ number of codes to model all words in the vocabulary, the capacity of the model is hurt significantly, since each word is forced to share embeddings with $2 \times \sqrt{|V|} - 1$ words which share the first (row) code or the second (column) code with it.

Therefore, we propose a Hybrid-LightRNN (HLR) mechanism which assigns exclusive embedding vectors to important words. Since frequent words have a large impact on the overall performance, in HLR, embeddings of the most frequent K words are not shared with other words. That is, high frequent words have exclusive embedding vectors and all other words will adopt the LightRNN style to share column and row embeddings.

Intuitively, the word table for HLR can be represented as below:

$$A = \begin{bmatrix} D & \text{UNK} \\ \text{UNK} & L \end{bmatrix}$$

where the matrix D is a sparse diagonal matrix to which frequent words are assigned. $L \in \mathbb{Z}^{d_1 \times d_2}$ is a dense matrix learned through LightRNN learning algorithm. To fit V words into the table, the dimensions of D and L should satisfy $K + d_1 \times d_2 \geq |V|$. Following LightRNN [108], we set $d_1 = d_2$ in this work so that L is a square matrix.

Byte Pair Encoding (BPE)

Byte Pair Encoding (BPE) [44, 155] is introduced to address the difficulties of translating rare and out-of-vocabulary words in machine translation. BPE is of interest here since it can reduce the vocabulary size effectively so as to speed up the softmax computation.

In the encoding learned by the BPE, each code is a subword. Formally, BPE learns the code dictionary $\boldsymbol{S}$ as follows: we initialize the code dictionary as the set of all possible characters and break all words into sequences of codes. Then we iteratively run the following steps to add new codes to the dictionary:

1. Count the frequency of all code pairs within training data. Find out the most frequent pair/bigram of codes A and B.
2. Add the new code AB to the dictionary. Replace all occurrence of pair (A, B) with AB.
3. End the iteration if the dictionary size reaches a threshold. Otherwise go to step 1.

BPE is an algorithm based on heuristics. However, similar to our Hybrid-LightRNN method, the strong inductive bias of BPE always gives more capacity to frequent words since the more frequent words will be segmented into fewer parts, which will lead to more exclusive embeddings. For many languages with concatenative morphology, words end up largely segmented into common morphemes which allows their effective encoding based on shared semantics. Even "unseen" words are likely to be segmented into seen morphemes with meaningful embeddings.

3.4 Related Work

Apart from the previously mentioned related works, mixture of softmaxes is closely related to works that mix representation vectors [37, 157]. Yang et al. [194] show that this approach does not solve the softmax bottleneck problem. Hierarchical Softmax [125] is an extensively studied technique to improve the efficiency of softmaxes. Morin and Bengio [125] use the synsets in the WordNet to build the hierarchical tree. Mnih and Hinton [123] propose to learn the hierarchical tree with a clustering algorithm. Although hierarchical Softmax can reduce the time and memory consumption during training, it still requires computing the softmax over the whole vocabulary during testing. The idea of separately modeling frequent words is also explored in Adaptive Softmax [51]. Noise Contrastive Estimation [57, 124] and Negative Sampling [122] can also speed up softmax during training. Instead of optimizing the softmax computation process directly, in this Chapter, we mainly focus on efficient vocabulary representation methods.

3.5 Experiments

In this section, we describe our experiments on machine translation and image captioning and study our models quantitatively and qualitatively.

Experiment Settings

Machine Translation We first evaluate our models on the IWSLT 2014 German to English (DE-EN) dataset [14]. Following [27], we set the word-level vocabulary size as 30k for both English and German. For Hybrid-LightRNN, we set K (number of words having exclusive embedding vectors) to $9,652$ and set d_1 and d_2 to 174 to represent a total of 30k words. For a fair comparison, we learn a subword-level vocabulary with 10k merging operations for English and German separately. As model performances exhibit small variances on IWSLT, we run each experiment for five times with different random seeds and report the average performance and the standard deviation. We also test our best model on the standard WMT 2014 English-to-German (EN-DE) and English-to-French (EN-FR) benchmarks, consisting of $4.5\mathrm{M}$ and $36\mathrm{M}$ sentence pairs respectively. We follow the preprocessing steps mentioned in Luong et al. [116]. We employ BPE with $32\mathrm{k}$ merge operations for both tasks. The transformer model [178] is employed as our baseline. Our configuration largely follows the configuration of Vaswani et al. [178], except that we multiply the original learning rate by 0.8 for the transformer equipped with MoS. Specifically, we test the transformer base configuration, which has embedding of dimension 512, the dimension of the inner layer 2048 and the number of attention heads 8. We use the Adam optimizer [81] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$. We set the mixture number to 9. We use the BLEU [133], METEOR[1] [32] and unigram Fmean[2] [100] to assess the translation quality. Our transformer training code is based on an open source toolkit THUMT [200]. All models are decoded with the beam search algorithm of beam size 4.

Image Captioning We conduct experiments on the MSCOCO dataset [111] and follow the same preprocessing procedure and the train/validation/test split as used in Karpathy and Fei-Fei [71]. We use the Neural Image Caption (NIC) model [180] as the baseline model. Following Dai et al. [27], we employ a pretrained 101-layer ResNet [62] instead of a GoogLeNet to extract a feature vector from an input image. Following previous works [19, 27], we employ an LSTM of size 512 as the decoder and report BLEU-4, METEOR and CIDERr scores using the scripts provided by Chen et al. [19].

		Machine Translation (IWSLT)		
Model	# Softmaxes	BLEU	METEOR	Fmean (%)
Baseline	1	29.86 ± 0.22	26.99 ± 0.07	64.83 ± 0.07
HLR-MoS	9	30.93 ± 0.38	27.79 ± 0.07	65.69 ± 0.12
BPE-MoS	9	**33.74** ± 0.18	**29.67** ± 0.13	**67.62** ± 0.06

Table 3.1: Overall performance comparisons on IWSLT'14 DE-EN. HLR denotes Hybrid-LightRNN. For all BLEU, METEOR, Fmean, the higher, the better.

Main Results

In our experiments, we denote Hybrid-LightRNN-MoS and BPE-MoS as the seq2seq models with MoS which employ Hybrid-LightRNN and BPE respectively. The baseline seq2seq model with word-level vocabulary is denoted as Baseline.

[1] `https://www.cs.cmu.edu/~alavie/METEOR/README.html`

[2] Lavie et al. [100] show compared to unigram F1, Fmean has better correlation with human judgements.

Model	# Softmaxes	Image Captioning (MSCOCO) BLEU-4	METEOR	CIDEr
Baseline	1	29.64 ± 0.20	23.60 ± 0.12	88.50 ± 0.47
Hybrid-LightRNN-MoS	9	30.02 ± 0.16	23.87 ± 0.18	88.96 ± 0.21
BPE-MoS	9	**30.06 ± 0.10**	**24.00 ± 0.24**	**89.26 ± 0.11**

Table 3.2: Overall performance comparisons on MSCOCO.

Overall Performances on IWSLT and MSCOCO We show the comparison between a word-level vocabulary seq2seq model with the Hybrid-LightRNN-MoS and BPE-MoS in Tables 3.1 and 3.2. Hybrid-LightRNN-MoS and BPE-MoS both outperform the baseline on both tasks. Specifically, on machine translation, BPE-MoS can significantly outperform the baseline and Hybrid-LightRNN. We will further analyze this finding in the following section.

Model	**WMT'14 EN-DE**			**WMT'14 EN-FR**		
	BLEU	METEOR	Fmean	BLEU	METEOR	Fmean
Our Transformer	27.4	48.2	60.4	39.0	58.1	65.6
Transformer-MoS	28.1	48.7	60.8	39.8	58.7	66.1

Table 3.3: Experiment results on the WMT 2014 English-German (EN-DE) and English-French (EN-FR) where Transformer-MoS denotes the Transformer model with MoS and BPE-based vocabulary. MoS-based models are significantly better than their corresponding baselines ($p<0.05$).

Performances on WMT 14 EN-DE and EN-FR Since BPE is significantly better than Hybrid-LightRNN and Baseline, we only test BPE-MoS on WMT. As shown in Table 3.3, we achieve 28.1 and 39.8 BLEU scores respectively on WMT 14 EN-DE and EN-FR, improving the transformer model by 0.7 and 0.8 BLEU scores.

Memory and Time Efficiency We study the memory consumption and efficiency of Hybrid-LightRNN-MoS and BPE-MoS. As shown in Table 3.4, when applying to the Baseline model and the MoS model, BPE and Hybrid-LightRNN can reduce the time and memory usage with no performance losses. When there are more mixtures, the efficiency improvements will continue to grow since computing softmaxes take a larger proportion of time.

Comparisons between BPE and Hybrid-LightRNN We see that BPE is significantly better over all metrics with the similar training speed and memory on the machine translation task.

# Softmaxes	Model	Machine Translation (IWSLT)				
		Memory↓	Speed ↑	BLEU	METEOR	Fmean (%)
1	Baseline	8.32	8.69	29.86 ± 0.22	26.99 ± 0.07	64.83 ± 0.07
	HLR	4.12	**9.34**	29.89 ± 0.20	26.87 ± 0.04	64.74 ± 0.08
	BPE	**4.01**	9.21	32.81 ± 0.22	29.24 ± 0.11	67.13 ± 0.19
3	MoS	22.7	5.51	30.74 ± 0.17	27.47 ± 0.06	65.24 ± 0.10
	HLR-MoS	8.03	**8.34**	30.63 ± 0.24	27.54 ± 0.08	65.37 ± 0.14
	BPE-MoS	**7.91**	8.14	33.42 ± 0.14	29.52 ± 0.10	67.46 ± 0.12

Table 3.4: Memory and time efficiency comparisons on IWSLT DE-EN when using the same number of Softmaxes. Bold faces highlight the best in the corresponding category. The shown memory is in GB and the speed is in # update_steps/s. For all BLEU, METEOR and Fmean, the higher the better.

# Softmaxes	Model	High (7269 word types)	Rare (1223)	OOV (991)
1	HLR	46.01 ± 0.06	1.32 ± 0.15	0.0
	BPE	48.17 ± 0.43	18.53 ± 0.98	12.60 ± 0.85
3	HLR-MoS	47.56 ± 0.37	2.26 ± 0.24	0.0
	BPE-MoS	49.01 ± 0.26	18.18 ± 0.53	16.99 ± 0.69

Table 3.5: Unigram accuracy of words with various frequencies on IWSLT DE-EN. HLR denotes Hybrid-LightRNN.

To understand it, we show the unigram accuracy with respect to the word frequency. More specifically, we split words in IWSLT test set into three categories, high frequency words (words occur more than 5 times in the training set), rare words (words occur less than or equal to 5 times in the training data) and Out-of-Vocabulary words (words not in the training set but test set).

The result is shown in Table 3.5. We see that Hybrid-LightRNN and BPE can achieve similar unigram Fmean scores for high frequency words. However, when coming to rare and OOV words, the advantage of BPE becomes clear. Therefore, subword-based vocabulary is able to mitigate the open-vocabulary issue of the word-level vocabulary but Hybrid-LightRNN is not capable of handling rare or OOV words.

We further check OOV words which can be translated correctly by BPE and show some words in Table 3.6. These words are compounding words, numbers, names, etc. Although these words do not occur in the training set, their subwords do and BPE is able to translate these subwords to compose the correct word.

Word	BPE
non-image	non-@@ image
580,000	5@@ 8@@ 0,000
50-letter	50-@@ letter
imploded	imp@@ lo@@ ded
neuromarketing	neuro@@ marketing

Table 3.6: Some OOV words which can be translated correctly by the BPE-based model.

Analysis

In this section, we perform extensive studies to better understand our models.

Number of Softmaxes Since a larger mixture number would lead to a higher rank log probability matrix, we verify whether a larger mixture number leads to better performance. We vary the number of softmaxes in the BPE-MoS model and compare their performances on MT. As shown in Figure 3.2, more softmax components clearly lead to better performances. However, the improvement margin exhibits a diminishing return effect, which means that several softmaxes are enough to learn a high-rank matrix.

Mapping Table	Table Size	Learned Table	BLEU
Hybrid-LightRNN	10k	Yes	30.07
Hybrid-LightRNN	5k		29.69
Hybrid-LightRNN	1k		28.73
LightRNN	0.2k	Yes	27.39
Frequency table	0.2k	No	25.84
Random table	0.2k		24.98

Table 3.7: Average validation BLEU on IWSLT of Hybrid-LightRNN using different mapping tables.

Hybrid-LightRNN Ablation Study We further study the importance of the learned table and the importance of the model's capacity in Hybrid-LightRNN. Firstly, we vary the dictionary size to investigate whether it is necessary to give enough capacity to frequent words.

As shown in Table 3.7, larger dictionary sizes consistently lead to better performances. Secondly, when compared with LightRNN, Hybrid-LightRNN achieves an improvement of 2.68

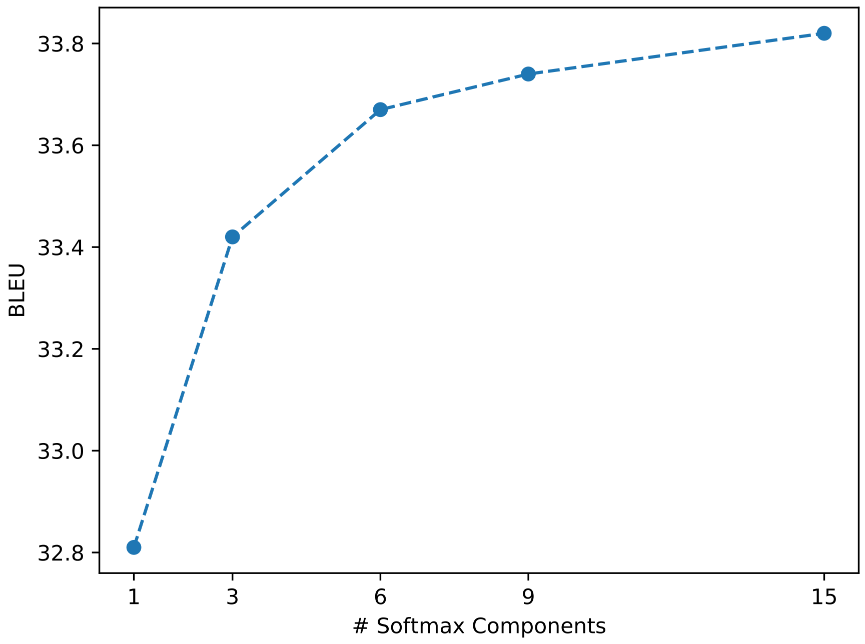


Figure 3.2: BPE-MoS's average performance over multiple runs on IWSLT DE-EN with various numbers of mixture components.

BLEU score, which shows that it is necessary to employ extra capacities for frequent words. Thirdly, as a sanity check of whether the table learning is necessary, we compare the table learned by LightRNN with the table obtained by simply sorting words based on their frequency and the table with random word allocations. The table learned by LightRNN outperforms models with the random table or the frequency-based table by BLEU scores of 2.41 and 1.55 respectively, which means that optimizing a language modeling objective learns an effective encoding function.

Mapping Table Qualitative Study In Hybrid-LightRNN, words in the same column/row share the same column/row embedding vector. Intuitively, it is important to group semantically-similar or syntactically-similar words into the same column/row. We examine whether the learned table has this property in Table 3.8. We find that most words within the same row are either semantically-similar or syntactically-similar to each other.

row	words							
45	700	3.3	28	19	7	86	35	...
48	around	between	by	into	down	for	off	...
54	mined	imaged	advised	pickled	outfitted	filled	withheld	...
91	bristol	chinatown	rochester	kingston	guangdong	guangzhou	chongqing	...
93	pursuing	posing	proposing	reacting	replacing	blogging	pointing	...

Table 3.8: Example mapping table where the row denotes the first code and the column denotes the second code. Numbers and places are grouped together in row 45, 91. Syntactically similar words are also grouped together in row 48, 54 and 93.

3.6 Summary

In this work, we investigate two algorithms, i.e., Byte Pair Encoding and Hybrid-LightRNN, to efficiently encode words so as to improve the memory- and time-efficiency of language generation models. BPE-based models achieve the best performance-efficiency trade-off due to its strong ability to handle rare and OOV words. Further, We demonstrate the effectiveness of Mixture of Softmaxes by improved performances on machine translation and image captioning. Since BPE is a heuristic-based word encoding mechanism which may be sub-optimal, it is meaningful to explore refine BPE in the future.

Chapter 4

Luna: Linear Unified Nested Attention

In this chapter, we will focus on the attention mechanism in transformer [178]. The quadratic computational and memory complexities of the transformer's attention mechanism have limited its scalability for modeling long sequences. To mitigate this issue, we propose Luna, a linear unified nested attention mechanism that approximates softmax attention with *two nested linear attention functions*, yielding only linear (as opposed to quadratic) time and space complexity. Specifically, with the first attention function, Luna packs the input sequence into a sequence of fixed length. Then, the packed sequence is unpacked using the second attention function. As compared to a more traditional attention mechanism, Luna introduces an additional sequence with a fixed length as input and an additional corresponding output, which allows Luna to perform the attention operation linearly, while also storing adequate contextual information. We perform extensive evaluations on three benchmarks of sequence modeling tasks: long-context sequence modeling, neural machine translation and masked language modeling for large-scale pretraining. Competitive or even better experimental results demonstrate both the effectiveness and efficiency of Luna compared to a variety of strong baseline methods including the full-rank attention and other efficient sparse and dense attention methods.

4.1 Introduction

Transformers [178] are surprisingly versatile models that perform well on a wide range of language and vision tasks, including machine translation [131, 178], language understanding [33], image recognition [35] and bioinformatics [119]. Attention [8] provides the key mechanism that captures contextual information from the entire sequence by modeling pairwise interactions between the inputs at every timestep. However, a common weakness of transformer is its quadratic time and memory complexity within the attention mechanism w.r.t. the length of the input sequence, which prohibitively restricts their potential application to tasks requiring longer input

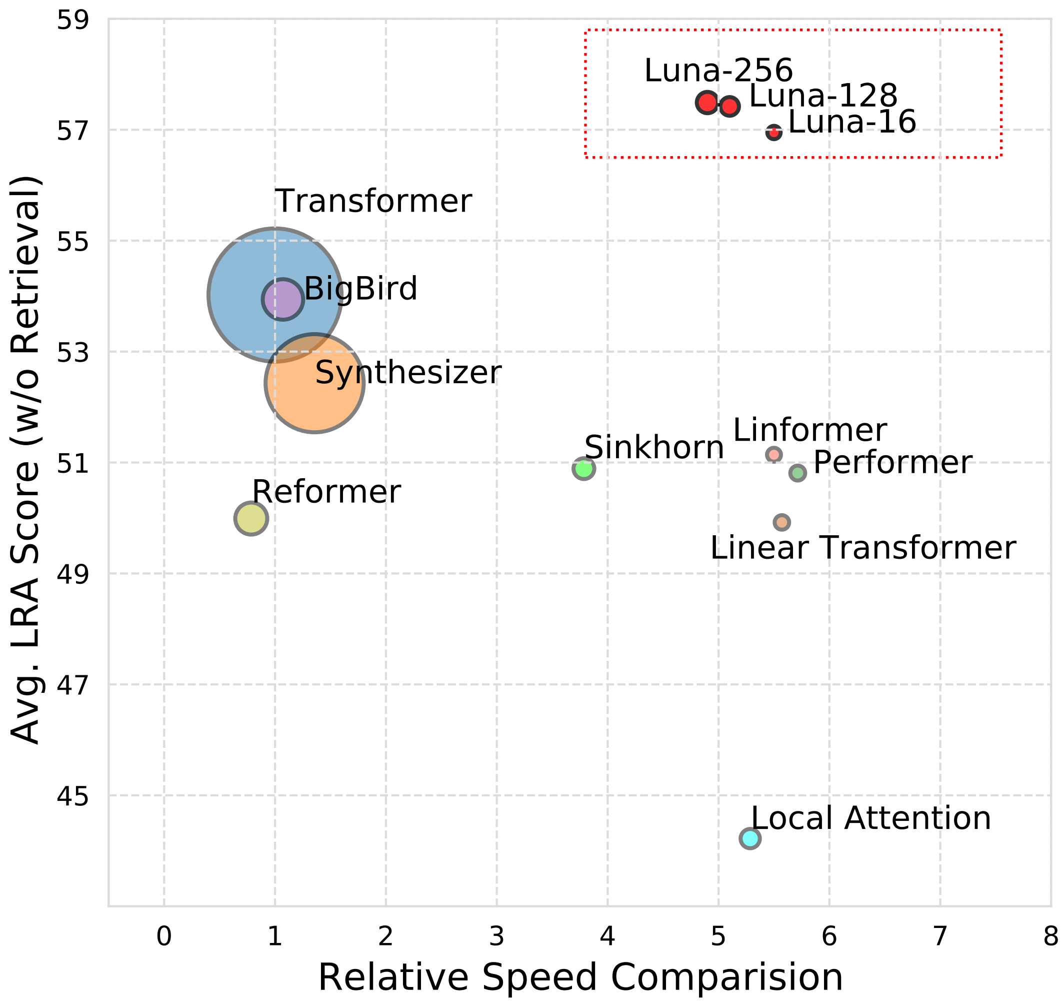


Figure 4.1: Trade-off between performance (y axis), speed (x axis) and memory (circle radius) on LRA benchmark [173].

sequences.

To mitigate this issue, a number of techniques have been recently introduced to improve the time and memory efficiency of transformer models (*'xformers'*) [172, 173]. One popular technique is using sparsity to restrict the attention field range, such as local attention [134], blockwise attention [139], strided attention patterns [10, 21], compressed attention [114], and attention with learnable patterns [86, 149, 171]. Another emerging approach is to improve efficiency by leveraging low-rank approximations of the attention matrix. Linformer [183], for example, projects the length dimension of key and value matrices to a fixed-dimensional representation by assuming low-rank structure in the full-rank attention matrix. Recently, some kernel-based methods, such as Linear Transformer [76], Performer [24] and Random Feature Attention [136], attempt to efficiently approximate regular (softmax) full-rank attention through kernelization. Although these models demonstrate better *asymptotic* complexity for long sequences, their performance remains behind the standard transformer with regular attention. Some of them do not support causal autoregressive decoding, which is required for neural machine translation. More discussions are mentioned in Tay et al. [172].

In this chapter, we propose *a linear unified nested attention* (**Luna**) mechanism, which uses two nested attention functions to approximate the regular softmax attention in transformer. Specifically, with the first attention function, Luna packs the input sequence into a sequence of fixed length. Then, the packed sequence is unpacked using the second attention function. As compared to a more traditional attention mechanism, Luna introduces additional sequence with a fixed length as input and an additional corresponding output. Importantly, the extra input allows Luna to perform attention operation linearly as efficiently as Linformer [183], while also storing adequate contextual information. Unlike Linformer, Luna is capable of modeling variable-length sequences and autoregressive (causal) attention. We perform extensive experiments on three sequence modeling tasks, including long-context sequence modeling, neural machine translation, and masked language modeling for large-scale pretraining and downstream task finetuning. Compared to a variety of strong baseline models, Luna achieves competitive or even better performance, while acquiring prominent gains of efficiency in both speed and memory (see Figure 4.1). More importantly, Luna manages to obtain superior performance with small projection lengths such as 16.

4.2 Background

This section provides an overview of the regular attention mechanism [178].

Attention

The traditional attention mechanism is a function:

$$Y = \mathrm{Attn}(X, C) = \omega\left(\frac{XW_Q(CW_K)^T}{\sqrt{d}}\right) CW_V \tag{4.1}$$

where the attention function $\mathrm{Attn} : \mathcal{R}^{n \times d} \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^{n \times d}$ takes as inputs two sequences: the query sequence $X \in \mathbb{R}^{n \times d}$ with length n and the context sequence $C \in \mathbb{R}^{m \times d}$ with length m, and output one sequence $Y \in \mathbb{R}^{n \times d}$ with the same length n as the query X. d is the embedding dimension, and $W_Q,\ W_K,\ W_V \in \mathbb{R}^{d \times d}$ are three learnable parameters that project the input sequences into the space of query, key and value matrices: $Q = XW_Q,\ K = CW_K,\ V = CW_V$. ω is an activation function, e.g. the *softmax* function in regular attention. Note that the formulation in (4.1) is applicable to both *cross-attention* where C and X are the representations from transformer encoder and decoder, respectively, and *self-attention* where X and C are the same sequence ($X = C$). In practice, the multi-head variant of attention [178], which performs the attention function h times in parallel, is commonly used. Throughout this paper, we omit h for simplicity.

In particular, the matrix $A = \omega(\frac{QK^T}{\sqrt{d_k}}) \in \mathbb{R}^{n \times m}$ in (4.1) is called the attention matrix which specifies the alignment scores between every pair of tokens in sequences of queries X and contexts C. Calculating A takes $O(nm)$ time and space, which is quadratic with respect to the sequence length and becomes a significant bottleneck when processing long sequences.

Transformer Layers

The other three key components of transformer, besides attention, are position-wise feed-forward networks (FFN), layer normalization [5] and the residual connection [61]. Technically, the position-wise feed-forward layer operates on each position independently and layer normalization and the residual connection play a crucial role in stabilizing the training process. Each transformer layer can be expressed as:

$$\begin{aligned} X_A &= \mathrm{LayerNorm}(\mathrm{Attn}(X, C) + X) \\ X' &= \mathrm{LayerNorm}(\mathrm{FFN}(X_A) + X_A) \end{aligned} \tag{4.2}$$

where X and C are the two input sequences and X' is the output of the transformer layer. The transformer layer in Equation 4.2 adopts the original post-layer normalization architecture [33, 178] that places layer normalization after residual connection, rather than pre-layer normalization [179, 182].

4.3 Linear Unified Nested Attention (Luna)

Our goal is to design an efficient attention mechanism to solve the quadratic complexity problem of full attention. We first introduce the proposed *linear unified nested attention* mechanism, named *Luna attention*, and the architecture of each Luna layer. Then, we present the variant of Luna for causal attention, named *Luna causal attention*. Finally, we discuss the differences between Luna and two closely related models: Linformer [182], Set Transformer [103] and kernels-based attentions [24, 136].

Pack and Unpack Attention

The key idea behind Luna is to decouple the regular attention function in Equation 4.1 into two nested attention operations, both of which have linear efficiency. To achieve this, besides the original query and context input sequences, Luna introduces an extra input that is a sequence with a fixed (constant) length. With this extra input as the query sequence, Luna uses its first attention, named *pack attention*, to pack the context sequence into a fixed-length sequence. Formally, let $P \in \mathbb{R}^{l \times d}$ denote the extra input sequence with the fixed length l. The pack attention first packs C to Y_P with P as the query sequence:

$$Y_P = \mathrm{Attn}(P, C) \tag{4.3}$$

where $\mathrm{Attn}(\cdot, \cdot)$ is the regular attention function in Equation 4.1, $C \in \mathbb{R}^{m \times d}$ is the context sequence, and $Y_P \in \mathbb{R}^{l \times d}$ is the output of the pack attention, which is named the *packed context*. Since the length of P is a constant l, the complexity of pack attention is $O(lm)$, which is linear with respect to m.

To unpack the sequence back to the length of the original query sequence X, Luna leverages its second attention, named *unpack attention*:

$$Y_X = \mathrm{Attn}(X, Y_P) \tag{4.4}$$

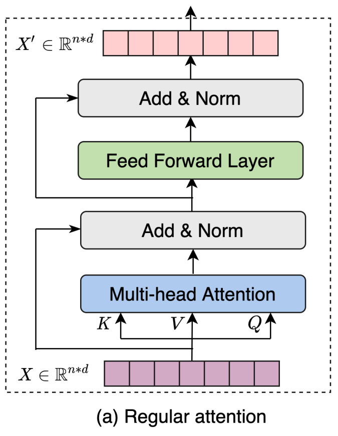
where $X \in \mathbb{R}^{n \times d}$ is the original query sequence. Similar to pack attention, the complexity of unpack attention is $O(ln)$, which is also linear with respect to n.

Encoding Contextual Information in P. The next question is where the extra input sequence P comes from. One straightforward choice is to format P as a learnable parameter of each Luna layer. One obvious drawback of this method, however, is that P would not capture any contextual information. To enhance the capacity of the Luna model, we propose to formulate Y_P as an additional output of each Luna layer, corresponding to P. Formally, the Luna attention function $\mathrm{LunaAttn}(\cdot, \cdot, \cdot)$ takes three sequences as input and generates two sequence as output:

$$Y_X, Y_P = \mathrm{LunaAttn}(X, P, C) \tag{4.5}$$

where the computation of Y_P and Y_X is in Equation 4.3 and Equation 4.4. By stacking multiple layers of Luna attention, the output Y_P from the previous layer, which captures contextual information of C, is employed as the input P of the next layer. For the first layer of Luna, we formulate P as learnable positional embeddings[1] [178].

[1]We also experimented with sinusoidal positional embeddings, and obtained similar results.



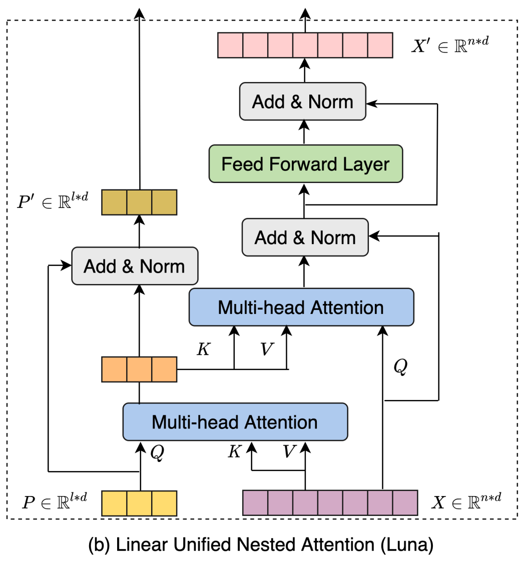


Figure 4.2: Illustration of the architecture of one Transformer encoder layer (left) versus one Luna encoder layer (right).

Reducing the Number of Parameters. Due to the two nested attention operations, there are two sets of parameters ($W_Q,\ W_K,\ W_V$) in a single Luna attention function. There are several techniques to reduce the number of parameters, such as parameter sharing [191]. In this work, we follow Wang et al. [183] to share W_K and W_V in each layer, and conduct experiments to analyze performance decline against Luna with full sets of parameters.

Luna Layers

The Luna attention is used as a drop-in-replacement for the regular attention. We incorporate the position-wise feed-forward network and layer normalization into Luna layers. Concretely, layer normalization is applied to both Y_X and Y_P, while FFN only to Y_X:

$$\begin{aligned} Y_X, Y_P &= \mathrm{LunaAttn}(X, P, C) \\ X_A, P_A &= \mathrm{LayerNorm}(Y_X + X),\ \mathrm{LayerNorm}(Y_P + P) \\ X', P' &= \mathrm{LayerNorm}(\mathrm{FFN}(X_A) + X_A),\ P_A \end{aligned} \tag{4.6}$$

where X' and P' are the two outputs of the Luna layer. The graphical specification of one Luna layer is illustrated in Figure 4.2.

Luna Causal Attention

As discussed in Tay et al. [172], the ability to support causal autoregressive decoding, i.e. attending solely to the past and current tokens, is required when designing efficient self-attention mechanisms. However, due to the pack attention that packs the long sequence X into a fixed (shorter) length, it is not straight-forward to support causal attention in Luna.

To design causal attention in Luna, we need to assume that the input P will not leak any future information of X to the history. Before we describe the Luna causal attention mechanism, we first define a causal function $f : \mathbb{R}^{n\times d_1} \times \mathbb{R}^{n\times d_1} \times \mathbb{R}^{n\times d_2} \rightarrow \mathbb{R}^{n\times d_2}$:

$$F \triangleq f(X, Y, Z), \text{ where } F_t = \frac{1}{t} X_t \sum_{j=1}^{t} Y_j^T Z_j \tag{4.7}$$

where $F \in \mathbb{R}^{n\times d_2}$ and F_t denotes the t-th row of F. From the definition of f in Equation 4.7, we see that F_t can only access the information of the past and present row of X, Y and Z.

To perform Luna causal attention, we first compute the attention matrix of the pack attention: $A_{pack} = \omega(PX^T/\sqrt{d})$. For simplicity, we omit the learnable parameters, e.g. $W_Q,\ W_K,\ W_V$ in Equation 4.1. Note that for A_{pack}, we cannot use the softmax function for ω, as the normalization term in softmax leaks future information of X to the history. Inspired by the causal attention mechanism in Linear Transformer [76], we use an activation function based on the exponential linear unit [25]: $\omega(\cdot) = \mathrm{elu}(\cdot) + 1$. With the causal function f in Equation 4.7, we compute the attention matrix of the unpack attention: $A_{unpack} = \omega(f(X, X, A_{pack}^T))$. Unlike A_{pack}, we can use $\omega(\cdot) = \mathrm{softmax}(\cdot)$ for A_{unpack}, because the normalization is along the l-dimension rather than the n-dimension of X. Finally, the output Y is computed by $Y = f(A_{unpack}, A_{pack}^T, X)$.

The complexity of the causal attention in Luna is still linear: $O(ln)$. One drawback of Luna causal attention, similar to the causal attention in Random Feature Attention (RFA) [136] and Linear Transformer [76], is its sequential computation for each timestep t. During inference time, our model will cache the current causal state to facilitate the decoding process.

Discussion

Relation to Linformer. One previous work closely related to Luna is Linformer [182]. Linformer linearly projects the context sequence $C \in \mathbb{R}^{m\times d}$ into a sequence with a fixed length l: $C' = EC$, where $C' \in \mathbb{R}^{l\times d}$ is the projected context sequence and $E \in \mathbb{R}^{l\times m}$ is the learnable projection matrix of each layer. Then, the attention operation is applied on the query X and the projected context C'. The pack attention in Luna is a generalization of the linear projection in Linformer. There are two main advantages to Luna over Linformer: i) with pack attention as the projection method, Luna is able to model sequences with various lengths. In contrast, Linformer requires the length of all input sequences to be the same m, due to the projection matrix

E, whose shape depends on m. ii) Luna achieves better expressiveness than Linear, not only due to the general projection method but also by encoding adequate contextual information into the projection via P (see Section 4.3). Experimental improvements over non-contextual projection demonstrate the effectiveness of Luna (see Section 4.4).

Relation to Set Transformer. The additional input P in Luna can be regarded as a side memory module that can access the entire sequence to gather contextual information. From this view of point, Luna is also closely related to Set Transformer [103], an early model to integrate a side memory module in Transformers. The major difference between Luna and Set Transformer is two-fold:

- Similar to the projection matrix in Linformer, the *inducing points* in Set Transformer are learnable parameters. Thus, these inducing points might be formulated as the non-contextual version of P in Luna. Experimental improvements over non-contextual projection demonstrate the effectiveness of Luna (see Section 4.4).
- Set Transformer is specifically designed for *set-input* problems which are permutation invariant. Therefore, they remove the positional encoding layer. However, in this work, the order information of tasks we are focusing on is important. For example, in the ListOps [127] task of the LRA dataset [173], the answer to the '[MAX 2 9 [MIN[4 7]]' is 9 but if we swap the 9 and 7, the answer will become 7. We argue that it is straightforward for Luna to model *set-input* problems through dropping the positional encoding layer.

Relation to kernel-based attention mechanisms. Another popular method to improve the efficiency of the attention mechanism is through kernelization [75, 136]. In the standard attention mechanism, the query and key inputs are multiplied together and passed through a softmax operation to form an attention matrix, which stores the similarity scores. Instead of explicitly computing this matrix, they decompose it back to a product of random nonlinear functions [143] of the original queries and keys. Furthermore, after decomposing the attention matrix, they can rearrange matrix multiplications to approximate the result of the regular attention mechanism, without explicitly constructing the quadratic-sized attention matrix. Since kernels are a form of approximation of the attention matrix, they can be also viewed as a type of low-rank approach. Their goal is to approximate the original attention matrix as much as possible by sampling random features. In Peng et al. [136], they find that random features sampling methods are crucial to the performance on various tasks showing the importance of a properly-chosen feature map. Instead of trying to approximate the attention matrix, we keep the original (softmax) attention mechanism in our two nested attention functions. Our goal is to first compress the contextual information into a compact contextual memory (P) via the pack attention from which we can obtain the final output through the unpack attention. The attention matrices in these two nested attentions are explicitly computed and no information will be lost. However, some contextual information may lose if we have a short P.

4.4 Experiments

Long-Context Sequence Modeling

We evaluate the effectiveness and efficiency of Luna on the Long Range Arena (LRA) benchmark recently introduced by Tay et al. [173], which is designed for the purpose of evaluating efficient transformer models under the long-context scenario. They collect five tasks in this benchmark which are ListOps [127], byte-level text classification (Text; 118), byte-level document retrieval (Retrieval; 140), image classification on sequences of pixels (Image; 93) and Pathfinder [112]. These tasks consist of input sequences ranging from 1K to 8K tokens and span across a variety of data types and modalities.

To ensure fair comparisons, for all tasks except for the task Retrieval, we closely follow the model configurations in Tay et al. [173] such as data preprocessing, data split, model architecture, etc. For the task of Retrieval, we find that models are not fully converged when being trained for 5K steps as stated in Tay et al. [173]. Therefore, we train models for 20K steps for this task and obtain much better results. For a direct comparison, besides the average performance of models across all tasks, we also report the average accuracy on tasks excluding Retrieval. We run each experiment five times with different random seeds and report the average accuracy with the standard deviation.

Results. The results of various models on the LRA benchmark are presented in Table 4.1. For our proposed method, we report results from models of three different projected dimensions (16, 128 and 256). First, we note that Luna achieves good results on all tasks consistently compared to the transformer model and significantly outperforms all the other baseline methods in terms of the average accuracy. By taking a closer look at the accuracy for each individual task, Luna wins over baseline models on three out of five tasks and performs comparably with the best performed model on the other two tasks, i.e. ListOps and byte-level text classification. Notably, Luna improves over the transformer model on image classification and pathfinder by a large margin. Second, we observe that although Luna achieves the best average performance with a projection dimension of 256, it also performs considerably well with smaller projection dimensions (16 and 128). This demonstrates the effectiveness of Luna even with small projected dimensions.

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg.	Avg. (w/o rtl)
Transformer	36.37	64.27	57.46	42.44	71.40	54.39	53.62
Transformer (re-impl)	37.11±0.2%	65.21±0.6%	79.14±0.4%	42.94±0.2%	71.83±0.6%	59.24	54.27
Local Attention	15.82	52.98	53.39	41.46	66.63	46.06	44.22
Sparse Trans.	17.07	63.58	**59.59**	44.24	71.71	51.24	49.15
Longformer	35.63	62.85	56.89	42.22	69.71	53.46	52.60
Linformer	35.70	53.94	52.27	38.56	76.34	51.36	51.14
Reformer	**37.27**	56.10	53.40	38.07	68.50	50.67	49.99
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	51.39	50.89
Synthesizer	36.99	61.68	54.67	41.61	69.45	52.88	52.43
BigBird	36.05	64.02	59.29	40.83	74.87	**55.01**	53.94
Linear Trans.	16.13	65.90	53.09	42.34	75.30	50.55	49.92
Performer	18.01	65.40	53.82	42.77	77.05	51.41	50.81
RFA	36.8	66.0	56.1	-	-	-	-
Luna-16	37.06±0.3%	66.00±0.1%	79.38±0.6%	46.39±0.3%	78.36±0.2%	61.44	56.95
Luna-128	37.34±0.4%	65.98±0.1%	79.55±0.4%	47.47±0.6%	**78.89**±0.5%	61.85	57.42
Luna-256	37.45±0.5%	**66.11**±0.2%	**79.56**±0.7%	**47.86**±0.6%	78.55±0.4%	**61.91**	**57.49**

Table 4.1: Experimental results on the long range arena (LRA) benchmark. For Luna, we explore three projected dimensions: 16, 128 and 256. 'Avg. (w/o rtl)' denotes the averaged accuracy over all tasks excluding Retrieval. The performance of previous works except RFA are from Tay et al. [173]. RFA scores are from Peng et al. [136].

Memory and Speed Efficiency. Luna employs two nested linear attention functions to reduce the time and memory complexity compared to the vanilla transformer attention. Here, we examine the speed and memory footprint of various models with varying input lengths (1K, 2K, 3K and 4K). Following Tay et al. [173], all models are evaluated on the byte-level classification task with the same batch size (32). The result is shown in Table 4.2.

Considering the memory efficiency, Luna with a projected dimension of 16 is highly memory-efficient, which is only 10% of the vanilla transformer at 4K input sequence length. With larger projected dimensions, i.e. 128 and 256, Luna requires more memory but is still competitive compared to other efficient transformer models. In terms of time efficiency, Luna-16 speeds up over the standard transformer by 1.2-5.5 times, varying by the sequence length. Compared to other efficient transformers, Luna-16 performs comparably with the fastest models, i.e. Performer and Linformer. Overall, our models achieve competitive advantages both in time- and memory-efficiency over other models, while attaining the best performance on the LRA benchmark (see Figure 4.1).

In addition, we plot the trade-off among memory, time and averaged LRA scores without task Retrieval in Figure 4.1. Models such as Linformer and Performer, have faster speed and small memory requirements with the sacrifice of performance. However, besides competitive time- and memory-efficiency, Luna models retain superior performance even with a small projected

Model	Steps per Second ↑				Peak Memory Usage (GB) ↓			
	1K	2K	3K	4K	1K	2K	3K	4K
Transformer	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00
Local Attention	1.1	1.7	3.2	5.3	0.49	0.29	0.19	0.14
Linformer	**1.2**	**1.9**	3.7	5.5	**0.44**	**0.21**	0.18	**0.10**
Reformer	0.5	0.4	0.7	0.8	0.56	0.37	0.28	0.24
Sinkhorn Trans	1.1	1.6	2.9	3.8	0.55	0.31	0.21	0.16
Synthesizer	1.1	1.2	2.9	1.4	0.76	0.75	0.74	0.74
BigBird	0.9	0.8	1.2	1.1	0.91	0.56	0.40	0.30
Linear Trans.	1.1	**1.9**	3.7	5.6	**0.44**	0.22	**0.15**	0.11
Performer	**1.2**	**1.9**	**3.8**	**5.7**	**0.44**	0.22	**0.15**	0.11
Luna-16	**1.2**	1.8	3.7	5.5	**0.44**	0.23	0.17	**0.10**
Luna-128	1.1	1.7	3.4	5.1	0.49	0.28	0.21	0.14
Luna-256	1.1	1.7	3.3	4.9	0.60	0.33	0.23	0.16

Table 4.2: Training speed and peak memory consumption comparison of different models on byte-level text classification with various input lengths (1K, 2K, 3K and 4K). The best model is in boldface.

dimension (l=16). Besides the training efficiency, decoding efficiency is reported in Table 4.3.

Model	Batch per Second ↑				Peak Memory Usage (GB) ↓			
	1K	2K	3K	4K	1K	2K	3K	4K
Transformer	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00
Linformer-128	1.7	2.6	3.8	5.1	0.66	0.38	0.22	0.14
Luna-16	**1.9**	**3.1**	**4.0**	**5.7**	**0.65**	**0.36**	**0.20**	**0.13**
Luna-128	1.6	2.6	3.5	4.9	0.67	0.38	0.22	0.14
Luna-256	1.3	2.2	2.9	4.0	0.68	0.39	0.24	0.14

Table 4.3: Decoding speed and peak memory consumption comparison of different models on byte-level text classification with various input lengths (1K, 2K, 3K and 4K). The best model is in boldface.

All models are tested on a single GPU with batch size 16. Our Luna-16 model achieves the best efficiency performance. Linformer [183] with the projected length 128 has slightly better efficiency scores compared to Luna with the same projected length (Luna-128). This is because Linformer just have one project function but Luna has two nestedd attention functions. However, as discussed above, our Luna model has superior performance and is capable of modelling causal attention.

Models	Method	ListOps	Text	Retrieval	Avg.
Luna-16	[CLS]	36.96	64.25	78.93	60.05
	Y_P	37.31	64.30	79.22	**60.27**
Luna-128	[CLS]	37.13	64.38	79.15	60.22
	Y_P	37.42	64.60	79.61	**60.54**
Luna-256	[CLS]	37.25	64.57	79.29	60.37
	Y_P	37.63	64.54	79.74	**60.64**

Table 4.4: Performance comparison of two sentence representation methods on LRA benchmark.

Contextual information in P of Luna. Recently, a popular method to model the classification task using transformer-based models is to prepend a special symbol, [CLS], to every input example. The last hidden state of this symbol is regarded as the aggregate sequence representation. In Luna, we introduce an extra model input P which not only allows us to efficiently compute the attention mechanism but learn contextual information as well. Theoretically, the output of P, Y_p, is capable of learning the representation of the input sequence. To validate this, we extract the output of P, Y_P at the last layer and employ the mean pooling strategy over positions to obtain the final feature for classification. We test its performance on three long-text modeling tasks in LRA [173], i.e., ListOps, Text and Retrieval and report results in Table 4.4. We find that Y_P-based methods obtain better scores across all tasks against the [CLS]-based one, validating the powerful ability of p to encode contextual information of the input sequence.

Masked Language Modeling for Large-Scale Pretraining[2]

One popular application of transformer is to pretrain a large-scale language model on a large amount of data which can then be fine-tuned on a wide range of downstream tasks, such as BERT [33], RoBERTa [115], etc. Therefore, following Devlin et al. [33], we pretrain a Luna-128-based language model with BERT-base configuration on BookCorpus [203] and English Wikipedia as our pretraining set (3300M tokens in total) with the masked-language-modeling

[2]The pretraining experiment is done by Sinong at Facebook AI Research and finetuning part is done by Xiang.

(MLM) objective, on 64 Tesla V100 GPUs with 250K updates. We compare our models with RoBERTa-base, BERT-base and Linformer which are trained on the same training data.

Finetuning Luna After obtaining the pretrained Luna-based language model, we finetune it on various natural language processing tasks, including sentiment classification (SST-2; 161), natural language inference (QNLI; 144), textual similarity (QQP; 20 and question answering (RACE [98] and CommonsenseQA (CSQA; 170). For GLUE tasks, following Liu et al. [115], we consider a limited hyperparameter sweep for each task, with batch sizes $\in \{16, 32\}$ and learning rate $\in \{5e^{-6}, 1e^{-5}, 2e^{-5}\}$, with a linear warmup for the first 6% of steps followed by a linear decay to 0. Finetuning is performed for 10 epochs with early stopping based on each task's evaluation metric on the dev set. For QA tasks, we concatenate each candidate answer with the corresponding question and passage. We then encode every candidate and pass the [CLS] output at the last layer through a fully-connected layer, which is used to predict the correct answer. We truncate question-answer pairs that are longer than 128 tokens and, if needed, the passage so that the total length is at most 512 tokens. Following Liu et al. [115], we try a small range of possible values for hyperparameters, i.e., batch size $\in \{16, 32\}$, learning rate $\in \{1e^{-5}, 2e^{-5}, 3e^{-5}\}$ and dropout $\in \{0.1, 0.2, 0.3\}$.

	GLUE			**QA**	
Model	SST-2	QNLI	QQP	RACE	CSQA
BERT-base [33]	92.7	88.4	89.6	64.2	**53.3**
RoBERTa-base [115]	**93.1**	90.9	**90.9**	**65.6**	-
Linformer [183]	92.4	90.4	90.2	-	-
Luna-128 (Ours)	**93.1**	**91.2**	90.8	65.2	53.1

Table 4.5: Performance of various models on development set of benchmark natural language understanding tasks. Bold face indicates best performance.

The result is reported in Table 4.5. We observe that our Luna model has similar or slightly better downstream results compared to other pretrained language models. On QNLI and SST-2, Luna models obtain the best performance among all models, reaffirming the effectiveness of Luna in pre-training. This demonstrates the strong ability of Luna for language representations.

Machine translation

To evaluate Luna on sequence-to-sequence modeling, we conduct experiments on two standard machine translation benchmarks, i.e. WMT14 English-German (EN$\rightarrow$DE) (4.5M sentence pairs)

Model	WMT14 EN-DE			WMT14 EN-FR		
	BLEU	COMET	METEOR	BLEU	COMET	METEOR
Transformer-base	27.4	43.7	0.48	39.0	58.2	0.58
RFA [135]	27.0	43.1	0.48	38.6	56.2	0.57
Linear Transformer [76]	22.3	31.4	0.33	34.3	52.6	0.43
Luna ($l = 8$)	25.8	40.3	0.42	37.3	49.6	0.56
Luna ($l = 16$)	27.0	43.5	0.48	38.8	56.5	0.58
Luna ($l = 32$)	27.2	43.4	0.48	39.1	57.3	0.58

Table 4.6: Performance comparison on WMT14 EN$\rightarrow$DE and WMT14 EN$\rightarrow$FR.

and WMT14 English-French (EN$\rightarrow$FR) (36M sentence pairs). The data split and preprocessing steps follow those of Vaswani et al. [178], using the scripts from Fairseq [132]. The Luna models closely follow the architecture of transformer-base: 6 encoder and decoder layers with 8 attention heads and $d_{\text{model}}/d_{\text{hidden}} = 512/2048$. We train the transformer-base model with Adam [83]. We closely follow the training configurations mentioned in Vaswani et al. [178]. The translation quality is evaluated by BLEU [133] and METEOR[3] [32]. The checkpoint with the best validation loss is chosen as the final model. During inference, we use beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [190] for all models.

Results Table 4.6 presents the results of Luna along with standard transformer-base, Random Feature Attention (RFA) [136] and Linear Transformer [76] models. The number of features in RFA is 256. We also compare our model with Linear Transformer which uses a feature map based on the exponential linear unit activation [25]. For Luna, we report performance of models with three different projected lengths, i.e., 8, 16, 32. Luna with a small projected length (16 or 32) obtains similar performance to RFA with k = 256 feature maps. We see that Luna models with project lengths 16 and 32 achieve competitive results as the transformer-base model and no significant difference has been found via statistical significance test [91]. All these models outperform Linear Transformer which is consistent with findings in Peng et al. [136]. However, the Luna model with the projected length 8 has a clear performance drop compared to others. We attribute it to the lack of ability to pack contextual information for a shorter P.

Effect of Encoding Contextual Information into P. As discussed above, one advantage of Luna against Linformer is to incorporate contextual P by formulating it as an extra input. To

[3]`https://www.cs.cmu.edu/~alavie/METEOR/README.html`

Model	WMT'14 EN-DE		WMT'14 EN-FR	
	BLEU	METEOR	BLEU	METEOR
Non-Contextual	24.8	44.6	37.1	56.4
Contextual	27.0	47.7	38.8	58.0

Table 4.7: Performance comparison on WMT14 EN$\rightarrow$DE and WMT14 EN$\rightarrow$FR.

investigate the importance of this design, we conduct experiments on WMT'14 EN-DE and EN-FR to compare Luna with the baseline model where P is formulated as a non-contextual learnable parameter of each layer. For both the contextual and non-contextual models, we use $l = 16$. Table 4.7 lists the BLEU and METEOR scores on the development and test sets. Luna with contextual P significantly outperforms the baseline with non-contextual P, demonstrating the effectiveness of this design in Luna.

Inference Efficiency To verify that Luna has advantages over the standard transformer in terms of memory and latency during inference, we examine the speed and memory footprint of various models on WMT'14 EN-FR test set with respect to the number of tokens in the source sentence. All models are tested on a single GPU with batch size 1. The result is shown in Figure 4.3. In terms of memory consumption, we see that the memory consumption of the standard transformer increases quickly on longer sentences. However, the memory of Luna-based model is stable and the increasing ratio is smaller. If we aim to translate a sequence of length 80, the standard transformer requires around 50% extra memory than Luna models. Considering decoding speed, when translating shorter sentences, standard transformer achieves similar latency compared to Luna models. However, when sentences become longer, the speed advantage of Luna over the standard attention is clear. This observation is consistent with the goal of Luna, achieving better memory and speed efficiency when modelling long sequences. We also report BLEU scores of these models with respect to the length of source sentences on WMT'14 EN-FR test set. We find that these systems can achieve similar translation accuracy on various length buckets.

Contextual Information Compression in P

The packing attention aims to pack the context sequence into a fixed-length sequence, P. Typically, the length of the fixed-length sequence is much shorter than that of the input sequence. In this study, we would like to explore the effect of P with various lengths to encode the contextual information. Following [85], we conduct experiments on a synthetic task: duplicate a sequence of symbols. In this task, each training and testing example has the form $0w0w$ where

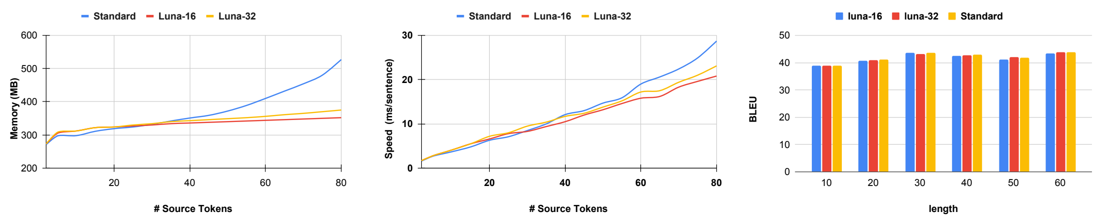


Figure 4.3: Decoding speed and peak memory consumption during inference on WMT'14 EN-FR test set with respect to the number of tokens in the source sequence. We also report the BLEU scores of various systems with respect to the source sentence lengths. All models are tested on the same GPU.

$w \in \{1, ..., N\}^*$ is a sequence of symbols ranging from 1 to N ($N = 127$). We show an example with the word w of length 3 here.

0	3	9	121	0	3	9	121

Figure 4.4: An example of the sentence duplicate task, The w is $< 3, 9, 121 >$ which is of length 3

In our experiment, each w is of length 511 (so the whole input sequence is of length 1024). The task is to train a language model, the goal of which is to predict the digit at the current position given all previous ones. Since each w is randomly generated, we just ask the model to predict the second half of the input (the second w).

This task can be done easily by the standard 2-layer transformer since the standard attention have access to all tokens and it is trivial for it to the position mapping. We employ a 2-layer Luna with model dimensions 256 and 4 attention heads with various projection lengths (32, 64, 128 and 256). All models are trained with 150k steps. The accuracy of these models is presented in Table 4.8.

This demonstrates that with large-enough projection lengths, Luna is able to pack the global context of the input sequence without loss of information. However, the large compression ratio (32) will cause severe contextual information loss.

4.5 Related Work

There has been significant prior work on improving the efficiency of Transformers, besides the two closely related works discussed in Section 4.3. The common techniques include, but are not

Input Length	Projected Length	Compression Ratio	Accuracy (%)
1024	32	32	55.2
1024	64	16	97.1
1024	128	8	99.9
1024	256	4	100

Table 4.8: Performance of Luna with various project lengths on the sequence duplication task.

limited to, weight sharing [29], quantization [39, 158], sparse attention [86, 134], and low-rank or compressed context [3, 103, 182]. In this section, we briefly review some recently proposed methods. For a detailed overview we refer the readers to Tay et al. [172].

Kernel Methods. A recently popular method to improve the efficiency of Transformers is to avoid explicitly computing the $m \times n$ attention matrix A in Section 4.1 by re-writing it with kernels. Typical models leveraging kernelization are Linear Transformer [76], Performer [24] and Random Feature Attention [136]. Since kernels are a form of approximation of the attention matrix, they can be also viewed as a form of low-rank method that compresses the context to a shorter length, such as Linformer [182] and the proposed Luna model.

Recurrence. The simplest technique to reduce the complexity of Transformer is to chunk input sequences into fixed blocks, with the obvious disadvantage of losing contextual information from past chunks. Transformer-XL [28] proposed a natural extension to the blockwise method to connect these blocks via a recurrence mechanism. Compressive Transformer [141] further extends Transformer-XL by maintaining a fine-grained memory of past chunk activations, which are discarded in Transformer-XL. Technically, Luna can be adapted to a recurrence method, by simply using P as an inherent memory module to maintain the recurrence across segments.

Alternative Architectures A considerable amount of effort has gone into designing Transformer alternatives such as MLP Mixers [175], G-MLP [113] and Flash [65]. Recently, Structured State Spaces [54]-based models achieve better very promising tasks on various tasks and surpass our proposed model. We recommend Tay et al. [172] to keep pace with the rate of innovation.

4.6 Summary

We have introduced Luna, a simple, efficient and effective linear attention mechanism used as a drop-in substitute for regular attention in transformer. By introducing an extra input with the fixed length, Luna is capable of capturing adequate contextual information while performing attention operations linearly. On three sequence modeling tasks, i.e., long-context sequence modeling, neural machine translation, and large-scale pretraining and finetuning, Luna achieves comparable or even better performance than a variety of strong baselines, while acquiring prominent gains of efficiency in both speed and memory. In future work, we are interested in combining Luna with recurrence methods where P can be used as a running memory across segments of inputs. Besides tasks explored in this chapter such as sentiment analysis, retrieval, machine translation and image classification, thanks to the strong ability of encoding contextual representation, it is easy to adopt Luna to other tasks such as summarization, image generation, etc.

4.7 Appendix

Long-Context Sequence Modelling

Tasks	LR	Dropout	Attn-Dropout
ListOps	1e-4	0.1	0.1
Text	5e-5	0.3	0.3
Retrieval	5e-5	0.1	0.1
Image	5e-3	0.1	0.3
Pathfinder	1e-3	0.2	0.1

Table 4.9: Hyperparameters of models in LRA tasks. LR and Attn-Dropout denote the learning, batch size and attention dropout.

For all tasks except Retrieval, we closely follow the model configurations in Tay et al. [173] such as data preprocessing, data split, model architecture, batch size etc. To guarantee convergence, we train models for the Retrieval task with 20k steps instead of the 5k steps prescribed inTay et al. [173]. The hyperparameters of models in these tasks are listed in Table 4.9. We mainly tune three hyperparameters: learning rate, dropout and attention dropout. For the other main hyperparametrs such as batch size, number of layers and number of warmup steps, we follow the guidance of Tay et al. [173].

Part II

Neural Machine Translation with High Decoding Parallelizability

Chapter 5

Semi-autoregressive Neural Machine Translation with Local Translation Mechanism

In this chapter, we start to focus on improving the inference latency via breaking the autoregressive property of NMT's traditional decoding method. A novel mechanism which is called Local Autoregressive Translation (LAT) is proposed to take the local dependency into account. During decoding time, LAT generates a short sequence autoregressively at each position. In global, these short sequences at all translation positions are generated in parallel so as to accelerate the whole translation process and improve the decoding parallelism. After obtaining a list of these short sequences, we further design a simple and effective merging algorithm to obtain the final translation output.

5.1 Introduction

Traditional neural machine translation (NMT) models [6, 22, 45, 168, 178] commonly make predictions in an incremental token-by-token way, which is called autoregressive translation (AT). Although this strategy can capture the full translation history, it has relatively high decoding latency due to the sequential property of its decoding algorithm. To make the decoding more efficient, non-autoregressive translation (NAT) [55] is introduced which generates multiple tokens in parallel instead of one-by-one. Although the decoding speed has been improved greatly, the translation quality encounters much decrease due to the conditional independence assumption.

In this chapter, we propose a semi-autoregressive NMT model with sub-linear parallel time generation. Specifically, we introduce a novel mechanism, i.e., local autoregressive translation (LAT), to take local target dependencies into consideration. At inference time, for a decoding position, instead of generating one token, we predict a short sequence of tokens (which we call a translation piece) for the current and next few positions in an autoregressive way. Globally, the short sequence at each decoding position is generated in parallel. Therefore, our model can effectively improve the parallelism of the conventional decoding algorithm.

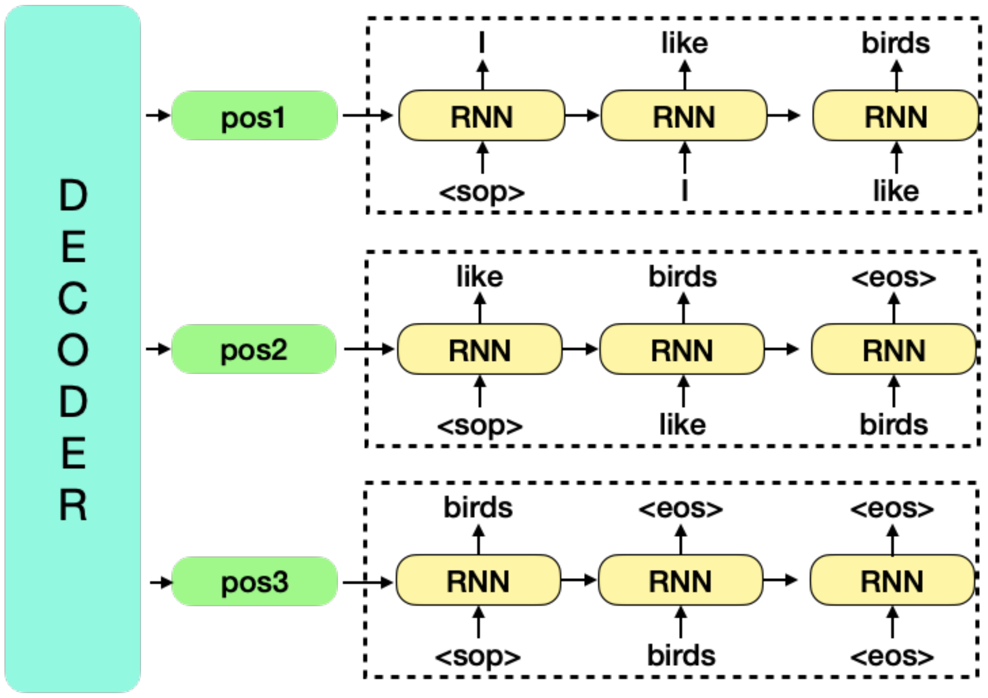


Figure 5.1: An example of the LAT mechanism. For each decoding position, a short sequence of tokens is generated in an autoregressive way. Short sequences at distinct translation positions are generated in parallel. $\langle$sop$\rangle$ is the special start-of-piece symbol. 'pos*' denotes the hidden state from the decoder at that position.

A simple example is shown in Figure 5.1. From it, we can find that with this mechanism, there can be overlapping tokens between nearby translation pieces. We take advantage of these redundancies, and apply a simple algorithm to align and merge all these pieces to obtain the full translation output. Specifically, our algorithm builds the output by incrementally aligning and merging adjacent pieces, based on the hypothesis that each local piece is fluent and there are overlapping tokens between adjacent pieces as aligning points. Moreover, the final output sequence is dynamically decided through the merging algorithm, which makes the decoding process more flexible. This process is non-neural which is highly efficient.

Inspired by BERT [33] and CMLM [47], we train our proposed semi-autoregressive NMT using the conditional masked language model framework, and similarly adopt iterative decoding, where tokens with low confidence scores are masked for re-prediction in more iterations. With evaluations on five translation tasks, i.e., WMT'14 EN$\leftrightarrow$DE, WMT'16 EN$\leftrightarrow$RO and IWSLT'14

DE→EN, we show that our method could achieve similar or better performance compared with strong iterative NAT and standard NMT models while gaining nearly 2.5 and 7 times speedups, respectively. Experimental analysis shows that compared to the NAT model, our method is capable of effectively reducing repeated translations and performing better on longer sentences.

5.2 NMT with Local Autoregressive Translation (LAT)

Model

Our LAT-based semi-autoregressive NMT model is built upon bidirectional transformer encoder and decoder [33, 47] and a local translator. We adopt a lightweight LSTM-based sequential decoder as the local translator upon the transformer decoder outputs. For a target position i, the bidirectional transformer decoder produces a hidden vector pos_i, based on which the local translator predicts a short sequence of tokens in an autoregressive way, i.e., $t_i^1, t_i^2, ..., t_i^K$. Here K is the number of location translation steps, which is set to 3 in our main experiments to avoid affecting the speed much.

Training

Our model is trained under the conditional masked language modelling framework, which obtains the full target sequence through predicting the masked target tokens based on the source and unmasked target sequences. Given a pair of source and target sequences S and T, we first sample a masking size from a uniform distribution from [1, N], where N is the target length. Then this size of tokens is randomly picked from the target sequence and replaced with the ⟨mask⟩ symbol. We refer to the set of masked tokens as T_{mask}. Then for each target position, we adopt a teacher-forcing styled training scheme to collect the cross-entropy loss for predicting the corresponding ground-truth local sequences, the size of which is $K = 3$.

Assume that we are at position i, we simply setup the ground-truth local sequence $t_i^1, t_i^2, ..., t_i^K$ as $T_i, T_{i+1}, ..., T_{i+K-1}$, where T_i denotes the i-th token in the full target ground-truth sequence. Different from BERT [33] and CMLM [47] where they just focus on the masked token, we include all tokens in our final loss, but adopt different weights for the masked tokens that do not appear in the inputs. Therefore, our token prediction loss function is:

$$\begin{aligned}\mathcal{L} = &-\sum_{i=1}^{N}\sum_{j=1}^{K} \mathbb{1}\left\{t_i^j \in T_{mask}\right\} \log(p(t_i^j)) \\ &-\sum_{i=1}^{N}\sum_{j=1}^{K} \mathbb{1}\left\{t_i^j \notin T_{mask}\right\} \alpha \log(p(t_i^j))\end{aligned}$$

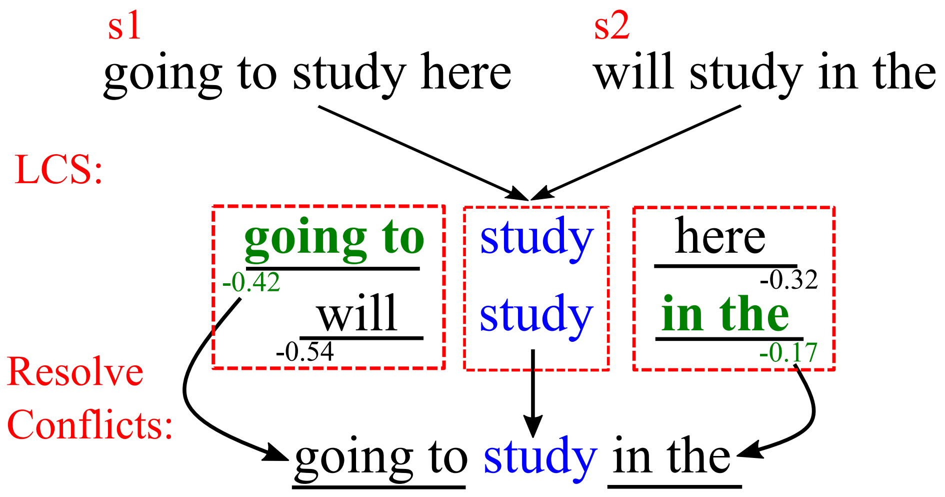


Figure 5.2: An example of merging two pieces of tokens.

Here, we adopt a weight α for the tokens that are not masked in the target input, which is set as 0.1 so that the model could be trained more on the unseen tokens. Lastly, since we need to determine how many translation positions beforehand at the decoding time, following CMLM [47], we add a target length predictor based on the source input representations which is trained with the translation objective together.

Decoding

During inference, a special token, $\langle\text{sop}\rangle$ (start of piece) is fed into the local translator to generate a short sequence based on the pos_i which is the hidden state from the bidirectional transformer decoder output at ith position. All short sequences have a fixed length which is consistent with our training process and allows for efficient parallel implementation. After generating the local pieces for all target positions in parallel, we adopt a simple algorithm to merge them into a full output sequence. This merging algorithm is described in detail in the next section. To refine the translation quality, we also perform iterative decoding following the same Mask-Predict strategy [33, 46]. In each iteration, we take the output sequence from the last iteration and mask a subset of tokens with low confidence scores by a special $\langle\text{mask}\rangle$ symbol. Then the masked sequence is fed together with the source sequence to the decoder for the next decoding iteration.

Following Ghazvininejad et al. [46], a special token LENGTH is added to the encoder, which is utilized to predict the initial target sequence length. Nevertheless, our algorithm can dynamically adjust the final output sequence during the merging process.

Algorithm 1: Merging two pieces.

```
Input: Two pieces of tokens: $s1$, $s2$.
Output: A merged sequence $s'$.
// Call Longest Common Subsequence
MatchedPairs = LCS($s1$, $s2$);
if MatchedPairs.size() == 0 then
    return $s1$+$s2$ ;                              // Simple concat
else
    $s'$ = [] ;                                      // Initialize
    $p1$, $p2$ = -1, -1 ;                            // Previous idxes
    // Add sentinel indexes.
    MatchedPairs += [($\infty$, $\infty$)];
    foreach $i1$, $i2$ in MatchedPairs do
        $span1$ = $s1$[$p1$+1:$i1$];
        $span2$ = $s2$[$p2$+1:$i2$];
        // Solve conflicts by scores.
        if score($span1$) $\geq$ score($span2$) then
            $s'$ += $span1$;
        else
            $s'$ += $span2$;
        // Align the matched ones.
        if $i1 \neq \infty$ then
            $s'$ += [align($s1$[$i1$], $s2$[$i2$])];
        $p1$, $p2$ = $i1$, $i2$;
    return $s'$;
```

Merging Algorithm

During decoding, the model generates local translation pieces for all decoding positions. We adopt a simple algorithm that incrementally builds the output through a piece-by-piece merging process. Our hypothesis is that if the local autoregressive translator is well-trained, then 1) the token sequence inside each piece is fluent and well-translated, 2) there are overlaps between nearby pieces, acting as aligning points for merging.

We first illustrate the core operation of merging two consecutive pieces of tokens. Algorithm 1 describes this procedure and Figure 5.2 provides an example. Given two token pieces $s1$ and $s2$, we first use the Longest Common Subsequence (LCS) algorithm to find matched tokens (Line

1). If there is nothing that can be matched, then we simply do concatenation (Line 3), otherwise we solve the conflicts of the alternative spans by comparing their confidence scores (Line 9-14). Finally we can arrive at the merged output after resolving all conflicted spans.

In the above procedure, we need to specify the score of a span. Through preliminary experiments, we find a simple but effective scheme. From the translation model, each token gets a model score of its log probability. For the score of a span, we average the scores of all the tokens inside. For aligned tokens, we choose the highest scores among them for the later merging process (Line 16).

With this core merging operation, we apply a left-to-right scan to merge all the pieces in a piece-by-piece fashion. For each merging operation, we only take the last K tokens of $s1$ and the first K tokens of $s2$, while other tokens are directly copied. This ensures that the merging process will only be local and mitigate the risk of wrongly aligned tokens. Here, K is again the local translation step size.

Although our merging algorithm is actually autoregressive, it does not include any neural network computations and thus can run efficiently. In addition to efficiency, our method also makes the decoding more flexible, since the final output is dynamically created through the merging algorithm. More details can be found in Appendix.

Comparing to Cascaded Text Generation [30] In their decoding process, the translation process is viewed as a conditional random field (CRF) [97] over a sequence of tokens. Given a m-th order CRF model, instead of employing traditional beam search, they propose an alternative cascaded decoding approach. Specifically, this algorithm is mainly based on iteratively computing max-marginals [187] for progressively higher-order models (from 0 to m-1) while filtering out unlikely spans based on the score of the “best” sequence with a given n-gram. Through this process, they can obtain high-fidelity translations. In our decoding process, we first directly generate m-gram (m here is the local translation steps) at each position, then merge them to a translated sequence. Our method to refine translation quality is to mask some tokens with lower confidence scores and re-predict them through the same local translation + merging processes. However, the refinement process in cascaded text generation happens in pruning unlikely candidates from low to high order models directly.

#	Model	Iterations	WMT'14 EN-DE	WMT'14 DE-EN	WMT'16 EN-RO	WMT'16 RO-EN	IWSLT'14 DE-EN	latency
1	AT	N	27.5/48.2	31.4/34.9	33.7/46.2	34.1/35.9	34.2/36.2	1.0×
2	CMLM	1	17.8/36.7	21.6/28.7	27.3/40.6	28.2/32.4	28.1/31.4	18.0×
3	LAT		25.2/46.1	29.9/33.8	30.7/42.6	31.2/33.8	31.9/33.7	15.7×
4	CMLM	4	25.5/46.8	29.5/34.9	32.5/45.7	33.2/35.5	32.9/34.6	6.8×
5	LAT		27.4/48.0	32.0/35.2	33.3/46.1	32.9/34.9	34.1/36.4	6.7×
6	CMLM	10	26.6/47.9	30.1/34.3	33.1/45.5	33.3/35.2	33.4/35.8	1.7×

Table 5.1: The comparisons (on BLEU/METEOR and decoding latency) of CMLM, LAT and AT models. CMLM denotes the Mask Predict model [47].

5.3 Experiments

Experimental Setup

Following previous works [47, 102], we evaluate our proposed method on five translation tasks, i.e., WMT'14 EN↔DE, WMT'16 EN↔RO and IWSLT'14 DE→EN. Knowledge distillation [55, 64, 77, 202] is used to train our models. We follow most of the hyperparameters for the transformer-base [178] configuration, i.e., 6 layers for encoder and decoder, 8 attention heads, 512 embedding dimensions and 2048 hidden dimensions. The LAT is an LSTM-based neural network of size 512. The number of the location translation step is set as 3, i.e., generating three tokens at every position. Finally, we average 5 best checkpoints according to the validation loss as our final model. During decoding, for AT models, we use beam search with a beam size of 4 and length penalty $\alpha = 0.6$. Please refer to the Appendix for more details of the settings.

The average decoding time per sentence with batch size 1, is employed to measure the inference speed. All models' decoding speed is measured on a single GPU. The BLEU [133] and METEOR [1] [32] are used to evaluate the translation quality.

Main results

The main results are shown in Table 5.1. Compared to the AT model, our proposed semi-autoregressive NMT with 4 translation iterations achieves competitive results and 7× speedup. We further use the `compare-mt` to conduct the statistical significance test and no significant

[1]`https://www.cs.cmu.edu/~alavie/METEOR/README.html`

difference is found on all translation tasks except the WMT'16 RO-EN task. On this task, we find that our translations are generally shorter than golden sequences (BP < 0.95) and we attribute it to the inaccurate predicted length from our length predictor.

Compared to a non-autoregressive model, CMLM [47], with the same number of decoding iterations (row 2 vs. 3 and row 4 vs. 5), LAT performs much better in terms of the translation accuracy with a small extra speed cost, especially when the iteration number is 1. Note that since our method is not sensitive to predicted length, we only take one length candidate from our length predictor instead of 5 as in CMLM. Furthermore, LAT with 4 iterations could achieve similar or better results than CMLM with 10 iterations (row 5 vs. 6) on all translation tasks but have a nearly 4$\times$ decoding speedup.

Analysis

	# local translation steps (K)				
	2	**3**	**4**	**5**	**6**
BLEU	32.9	33.8	34.4	34.5	34.2
latency (ms)	69	72	76	77	79

Table 5.2: The performance of LAT models with respect to the number of local translation steps on IWSLT'14 DE-EN test set.

On local translation step. We also explore the effect of the number of local translation steps (K) on the IWSLT'14 DE-EN dataset. The results are shown in Table 5.2. Generally, with more local translation steps, there can be certain improvements on BLEU but with an extra cost at inference time. This improvement becomes smaller when increasing the number of local translation steps.

To understand why our proposed model has superior translation accuracy compared to the NAT model with the same number of decoding iterations, we also conduct two ablation studies.

On repeated translation. We compute the n-gram repeat rate (nrr, what percentage of n-grams are repeated by certain nearby n-grams) of different systems on WMT'14 EN-DE test set and the result is shown in Table 5.3. The nrr of CMLM with one iteration is much higher than other systems, showing that it suffers from a severe repeated translation problem. Conversely, LAT can mitigate this problem thanks to the merging algorithm and the introduction of the local dependency.

Model	**Iteration**	**ngram repeat rate (%)**			
		1	**2**	**3**	**4**
CMLM	1	20.85	3.78	1.06	0.37
LAT		**4.89**	**0.42**	**0.05**	**0.00**
CMLM	4	3.97	0.14	0.03	0.02
LAT		**3.32**	**0.08**	**0.00**	**0.00**
CMLM	10	3.56	0.08	0.02	0.02
AT	N	3.27	0.05	0.00	0.00
Reference	-	2.49	0.03	0.00	0.00

Table 5.3: N-gram repeat rates of various models on WMT'14 EN-DE test set. 'AT' here is a transformer base translation model.

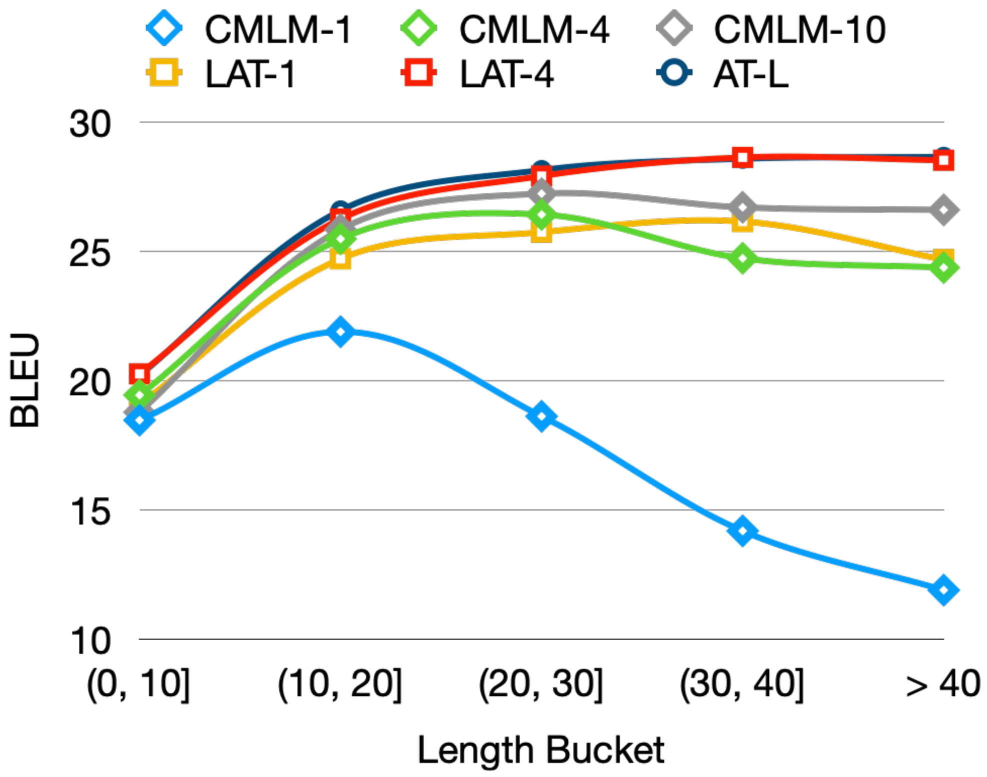


Figure 5.3: The BLEU scores of various systems with respect to the reference sentence lengths on WMT'14 EN-DE testset.

On sentence length. We explore how various systems perform on sentences with various lengths. The WMT'14 EN-DE test set is split into 5 length buckets by the source length. Figure 5.3 shows that our proposed semi-autoregressive NMT performs better than CMLM on longer sentences, which indicates the effectiveness of our methods at capturing certain target dependencies. For the non-autoregressive translation model with just 1 iteration, it performs poorly on translating long sentences. With LAT mechanism, 1 iteration-translation is able to obtain reasonable translations for those long sentences.

5.4 Summary

In this chapter, we design a semi-autoregressive NMT model with a novel mechanism called local autoregressive translation which can take the local dependency into consideration. Overall, our decoding method has the autoregressive attribute locally but non-autoregressive property in global. Specifically, short sequences are generated in parallel at all translation positions, then a simple algorithm is further designed to align and merge these short sequences. Experimental results show that our proposed model achieves competitive translation accuracy compared to the standard AT model with 7$\times$ speedup. In contrast to the non-autoregressive model, our model can effectively boost the translation quality with a small extra inference speed cost. Furthermore, given the similar performance, our model can achieve up to 4$\times$ speedup as compared to an iterative NAT model. In this chapter, all short sequences have a fixed length which is consistent with our training process and allows for efficient parallel implementation. For the purpose of higher decoding speed, our merging algorithm is deterministic but heuristic which maybe suboptimal. In the future, we would like to explore short sequences with dynamic lengths and more advanced merging algorithm.

5.5 Appendix

Preprocessing

We follow the standard pre-processing procedure in prior works [102, 178]. All datasets are segmented into subwords through byte pair encoding (BPE) [155]. The BPE code is learnt from the combination of source and target data for WMT datasets. For IWSLT, the bpe code is learned from the source and target data separately. Table 5.4 lists some details.

Optimization

We sample weights from $\mathcal{N}(0, 0.02)$, initialize biases to zero, and set layer normalization parameters to $\beta = 0$, $\gamma = 1$. For regularization, we use 0.3 dropout, 0.01 L2 weight decay, and

Dataset	Vocab. Size	Data size
IWSLT	10k	150k
WMT14 EN↔DE	32k	4.5M
WMT16 EN↔RO	40k	600k

Table 5.4: Pre-processing details of various translation benchmarks. Vocab. size denotes vocabulary size.

smoothed cross-entropy loss with $\epsilon = 0.1$. We train batches of 128k tokens using Adam [82] with $\beta = (0.9, 0.999)$ and $\epsilon = 10^{-6}$. The learning rate warms up to a peak of 5×10^{-4} within 10,000 steps, and then decays with the inverse square-root schedule. We train our models for 300k steps with batch size 128k [46] for WMT datasets. For the IWSLT dataset, we train our models for 50k steps with batch size 32k.

Model Parameter Size

The averaged size of parameters for all models are shown in Table 5.5. These three kinds of models have similar number of parameters. LAT models have the most number of parameters due to the LSTM-based local translator.

Model	Parameter size
AT	60M
CMLM	62M
LAT	64M

Table 5.5: Number of Parameters of different models.

Length Adjustment for Intermediate Iterations

Since our merging algorithm produces the output dynamically, the output length is usually not the same as the number of input pieces. In iterative decoding, we find it helpful to adjust the output sequence's length to the input length in intermediate iterations. This is achieved by adding or deleting the special $\langle mask \rangle$ symbols. Notice that for the final iteration, we do not apply any adjustments and keep the merged output sequence as it is.

For the length adjustment in the intermediate iterations, our goal is to adjust the output length of the merger (L_{out}) to be close to the input target length (L_{in}). If these two lengths are already equal or their relative difference is within a certain range (which is empirically set to 5%), we will do nothing. Otherwise, there can be two cases: 1) when L_{in} is larger than L_{out}, we further insert $L_{in} - L_{out}$ $\langle mask \rangle$ tokens into the sequence; 2) otherwise, we try to delete $L_{out} - L_{in}$ $\langle mask \rangle$ tokens. Notice that the addition or deletion operations happen after the masking procedure for the next iteration.

Here, we describe the addition case in detail. Suppose we need to further insert M masks into the output sequence, we decide the insertion places according to the position gaps. We adopt a simple position scheme for all the tokens. For each original token t_i^j (the j-th token in the i-th piece) in the input translation pieces, we set $i + j$ as its position. For each token in the output sequence after merging, since it can originate from multiple input tokens through aligning, we take the averaged value of all its source input tokens' positions. We calculate the position gap between each pair of nearby unmasked tokens in the output sequence and maintain a priority queue for all these gaps. Then we insert M masks once at a time. For each time, we select the current maximal gap, insert a $\langle mask \rangle$ to that position, and subtract that gap by 1. The case for deletion would be similar but in the opposite direction: select the minimal gap, delete one $\langle mask \rangle$ if there are any, and increase that gap by 1. We will delete nothing if there are no masked tokens in the selected gap.

Chapter 6

Fully Non-autoregressive Neural Machine Translation: Tricks of the Trade

In chapter 5, we introduce a semi-autoregressive NMT system with a local autoregressive translation mechanism, which can improve the parallelism of the traditional NMT decoding process. In this chapter, we aim for a full parallelism decoding process while maintaining superior translation accuracy at the same time.

6.1 Introduction

Non-autoregressive neural machine translation models [NAT, 55] attempt to generate output sequences in parallel to speed up the decoding process. The incorrect independence assumption nevertheless prevents NAT models from properly learning the dependency between target tokens in real data distribution, resulting in degraded performance compared to autoregressive (AT) models. One popular solution to improve the NAT translation accuracy is to sacrifice the speed-up by incorporating an iterative refinement process, through which the model explicitly learns the conditional distribution over partially observed reference tokens [48, 56]. However, recent studies [73] indicate that iterative NAT models seem to lose the speed advantage compared to AT models with careful tuning of the layer allocation. For instance, an AT model with the *deep encoder and shallow decoder* architecture obtains similar latency as iterative NAT models without hurting the translation accuracy.

Therefore, how to build a competitive fully NAT model without iterative refinements calls for more exploration. Several works [49, 138, 150] have recently been proposed to improve the translation quality of NAT, though the performance gap compared to the iterative ones remains. In this chapter, we first argue that the key to successfully training a fully NAT model is to perform *dependency reduction* in the learning space of output tokens (Section 6.2) from several aspects.

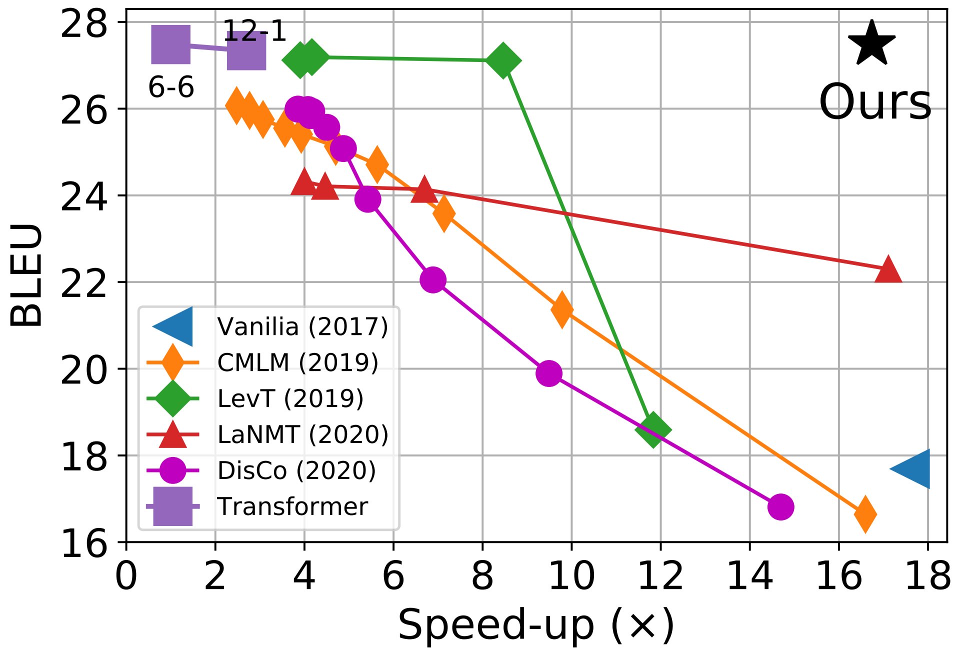


Figure 6.1: The translation quality v.s. inference speed-up of the proposed model with the AT (transformer) and existing popular iterative NAT models varying decoding iterations on WMT'14 En→De test set. The upper right corner achieves the best trade-off.

With this guidance, we revisit various methods which are able to reduce the dependencies among target tokens as much as possible including four different perspectives, i.e., training corpus, model architecture, training objective and learning strategy. The performance gap can not be near closed unless we combine these techniques' advantages.

We validate the proposed fully NAT model on standard translation benchmarks including 5 translation directions where our system achieves new state-of-the-art results for fully NAT models on all directions. We also demonstrate the quality-speed trade-off comparing with AT and recent iterative NAT models in Figure 6.1. Moreover, compared to the transformer baseline, our model achieves **16.5**$\times$ inference speed-up under the same software/hardware conditions while maintaining comparable translation quality.

6.2 Motivation

Given an input sequence $\boldsymbol{x} = x_1 \ldots x_{T'}$, an autoregressive model [6, 178] predicts the target $\boldsymbol{y} = y_1 \ldots y_T$ sequentially based on the conditional distribution $p(y_t|y_{<t}, x_{1:T'}; \theta)$, which tends to suffer from high latency in generation especially for long sequences. In contrast, non-autoregressive machine translation [NAT, 55], proposed for speeding-up the inference by gen-

erating all the tokens in parallel, has recently been on trend due to its parallelizable nature on devices such as GPUs and TPUs. A typical NAT system assumes a conditional independence in the output token space, that is

$$\log p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \sum_{t=1}^{T} \log p_\theta(y_t|x_{1:T'}) \tag{6.1}$$

where θ is the parameters of the model. Typically, NAT models are modeled with Transformer without causal attention map in the decoder side. As noted in Gu et al. [55], the independence assumption, however, generally does not hold in real data distribution for sequence generation tasks such as machine translation [148], where the failure of capturing such dependency between target tokens leads to a serious performance degradation in NAT. As shown in Figure 6.1, despite the inference speed-up, the vanilla NAT leads to a quality drop over **10** BLEU points.

To ease the modeling difficulty, recent state-of-the-art NAT systems [48, 56, 72, 102, 150, 160, 164] trade accuracy with latency by incorporating iterative refinement in non-autoregressive prediction. For instance, Gu et al. [56] learns to translate by editing (deletion, insertion) on previously generated sequence iteratively. Although iterative NAT models have already achieved comparable or even better performance than the autoregressive counterpart, Kasai et al. [73] shows that AT models with a deep encoder and a shallow decoder can readily outperform strong iterative models with similar latency, indicating that the latency advantage of iterative NAT has been overestimated.

By contrast, while maintaining a clear speed advantage, a fully NAT system – model makes parallel predictions with a single neural network forward – still lags behind in translation quality and has not been fully explored in literature [49, 109, 110, 117, 167]. This motivates us in this work to investigate various approaches to push the limits of learning a fully NAT model towards autoregressive models regardless of the architecture choices [73].

6.3 Methods

In this section, we discuss several essential ingredients to train a fully NAT model. As discussed in Section 6.2, we argue that the guiding principle of designing any NAT models is to perform *dependency reduction* as much as possible in the output space so that it can be captured by the NAT model. For example, iterative-based models [48] explicitly reduce the dependencies between output tokens by learning the conditional distribution over the observed reference tokens. The overall framework of training our fully NAT system is presented in Figure 6.2. We also summarize the pros/cons for each proposed method in Table 6.1 for reference.

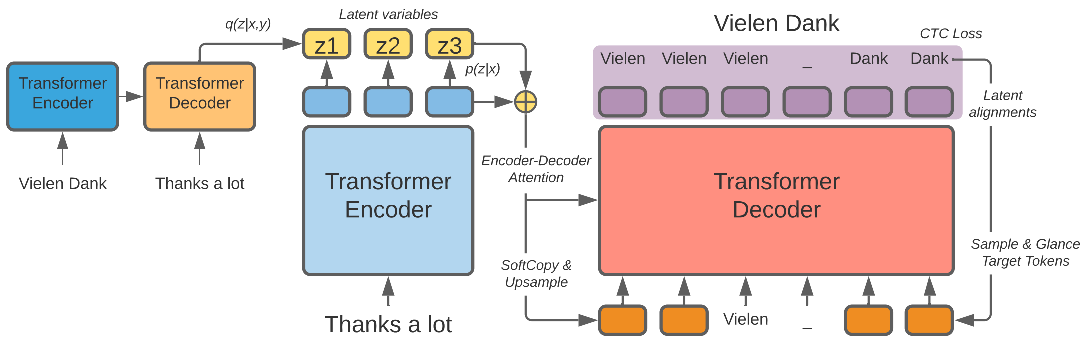


Figure 6.2: The overall framework of our fully NAT model.

Methods	**Distillation**	**Latent Variables**	**Latent Alignments**	**Glancing Targets**
What it can do?	simplifying the training data	model any types of dependency in theory	handling token shifts in the output space	ease the difficulty of learning hard examples
What it cannot?	uncertainty exists in the teacher model	constrained by the modeling power of the used latent variables	unable to model non-monotonic dependency, e.g. reordering	training / testing phase mismatch
Potential issues	sub-optimal due to the teacher's capacity	difficult to train; posterior collapse	decoder inputs must be longer than targets	difficult to find the optimal masking ratio

Table 6.1: Comparison between the proposed techniques for improving fully NAT models.

Data: Knowledge Distillation

The most effective *dependency reduction* technique is knowledge distillation (KD) [64, 78] which is firstly proposed to improve NAT in Gu et al. [55] and has been widely employed for all subsequent NAT models. The original target samples are replaced with sentences generated from a pre-trained autoregressive model. As analyzed in Zhou et al. [202], KD is able to simplify the training data where the generated targets have less noise and are aligned to the inputs more deterministically. Also, it shows that the capacity of the teacher model should be constrained to match the desired NAT model to avoid further degradation, especially for weak NAT students without iterative refinement.

Model: Latent Variables

Different from iterative NAT, *dependency reduction* can be done with (nearly) zero additional cost at inference by adding latent variables to the model. In such case, output tokens $y_{1:T}$ are modeled conditionally independent over the latent variables $\boldsymbol{z}$ which are predicted from the prior distribution:

$$\log p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \log \int_{\boldsymbol{z}} p_\theta(\boldsymbol{z}|\boldsymbol{x}) p_\theta(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}) d\boldsymbol{z} \tag{6.2}$$

z can be either extracted by a fixed external library (e.g. fertility in Gu et al. [55]), or jointly optimized with the NAT model using variational auto-encoders (VAEs) [70, 160] or normalizing flows [117].

In this work, we follow the formulation proposed in Shu et al. [160] where continuous latent variables $\boldsymbol{z} \in \mathbb{R}^{T' \times D}$ are modeled as spherical Gaussian at the encoder output of each position. Like typical VAEs [84], the model is trained by maximizing the evidence lower-bound (ELBO) with a posterior network q_ϕ:

$$\underbrace{\mathbb{E}_{\boldsymbol{z} \sim q_\phi} \left[\log p_\theta(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x})\right]}_{\text{likelihood}} - \mathcal{D}_{\text{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}) \| p_\theta(\boldsymbol{z}|\boldsymbol{x})) \tag{6.3}$$

where $\mathcal{D}_{\text{KL}}$ is the Kullback–Leibler divergence between the prior and posterior. In this work, we use a transformer to encode $q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$. Only the embedding layers are shared between θ and ϕ

Loss Function: Latent Alignments

Standard NMT models are trained with the cross entropy (CE) loss which compares the model's output with target tokens at each corresponded position. However, as NAT ignores the dependency in the output space, it is almost impossible for such models to model token offset accurately. For instance, while with little effect to the meaning, simply changing "*Vielen Dank !*" to "*, Vielen Dank*" causes a huge penalty for fully NAT models.

To ease such limitation, recent works propose to consider the latent alignments between the target positions, and optimize [49], or marginalize all alignments [110, 150]. As a special form of latent variables in loss computation, latent alignments can be easily computed through dynamic programming. In this work, we put our primary focus on Connectionist Temporal Classification (CTC) [52] as the latent alignments, considering its superior performance and the flexibility of variable length prediction. CTC is capable of efficiently finding all valid aligned sequences $\boldsymbol{a}$ which the target $\boldsymbol{y}$ can be recovered from, and marginalize log-likelihood:

$$\log p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \log \sum_{\boldsymbol{a} \in \Gamma(\boldsymbol{y})} p_\theta(\boldsymbol{a}|\boldsymbol{x}) \tag{6.4}$$

where $\Gamma^{-1}(\boldsymbol{a})$ is the collapse function that recovers the target sequence by collapsing consecutive repeated tokens, and then removing all blank tokens. Also, it is straightforward to apply the same CTC loss into the VAE models (Section 6.3) by replacing the likelihood term in Equation (6.3) with the CTC loss. Because of the strong assumptions of monotonic alignment, it is impossible to reduce all dependencies between target tokens in the real distribution.

Learning: Glancing Targets

Ghazvininejad et al. [48] show that it improves test time performance by glancing the reference tokens when training NAT models. That is, instead of $\log p_\theta(\boldsymbol{y}|\boldsymbol{x})$, we optimize $\log p_\theta(\boldsymbol{y}|\boldsymbol{m} \odot \boldsymbol{y}, \boldsymbol{x}), \boldsymbol{m} \sim \gamma(l, \boldsymbol{y}), l \sim \mathcal{U}_{|\boldsymbol{y}|}$, where $\boldsymbol{m}$ is the mask, and γ is the sampling function given the number of masked tokens l. As mentioned earlier, we suspect such explicit modeling of the distribution conditional to unmasked tokens assists the *dependency reduction* in the output space.

Naively applying random masks for every training example may cause a severe mismatch between training and testing. To mitigate this, Qian et al. [138] propose GLAT – a curriculum learning strategy, in which the ratio of glanced target tokens is proportional to the translation error of the fully NAT model. More precisely, instead of sampling uniformly, we sample l by:

$$l \sim g(f_{\text{ratio}} \cdot \mathcal{D}(\hat{\boldsymbol{y}}, \boldsymbol{y})) \tag{6.5}$$

where $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \log p_\theta(\boldsymbol{y}|\boldsymbol{x})$, $\mathcal{D}$ is the discrepancy between the model prediction and the target sequence, e.g. Levenshtein distance [105], and f_{ratio} is a hyperparameter to adjust the mask ratio. The original formulation [138] utilized a deterministic mapping (g), while we use a Poisson distribution to sample a wider range of lengths including "no glancing". Intuitively, a poorly trained model will glance at many target tokens. When the model becomes better and generates higher quality sequences, the number of masked words will be larger, which helps the model gradually learn generating the whole sentence.

The original GLAT [138] assumes to work with the golden length so that it can glance at the target by placing the target word embedding to the corresponded inputs, which is incompatible with CTC as we always require the inputs longer than the targets. To enable GLAT training, we glance at target tokens from the viterbi aligned tokens $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}\in\Gamma(\boldsymbol{y})} \log p_\theta(\boldsymbol{a}|\boldsymbol{x})$ which has the same length as the decoder inputs.

6.4 Experiments

We perform extensive experiments on three challenging translation datasets by combining all mentioned techniques to check (1) whether the proposed aspects for *dependency reduction* are complementary; (2) how much we can minimize the gap between a fully non-autoregressive model with the autoregressive counterpart.

Experimental Setup

Dataset and Preprocessing We validate our proposed models on three standard translation benchmarks with variant sizes, i.e., WMT14 English (EN) $\leftrightarrow$ German (DE) (4.0M pairs), WMT16 English (EN) $\leftrightarrow$ Romanian (RO) (610k pairs) and WMT20 Japanese (JA) $\rightarrow$ English (EN) (13M pairs after filtering). For EN$\leftrightarrow$DE and EN$\leftrightarrow$RO, we apply the same prepossessing steps and learn sub-words as mentioned in prior work (EN$\leftrightarrow$DE: 202, EN$\leftrightarrow$RO: 102). For JA$\rightarrow$EN, the original data (16M pairs) is first filtered with Bicleaner [151] [1] and we apply SentencePiece [94] to generate 32k subwords.

Knowledge Distillation Following previous efforts, we also train the NAT models on distilled data generated from pre-trained transformer models (*base* for WMT14 EN$\leftrightarrow$DE and WMT16 EN$\leftrightarrow$RO and *big* for WMT20 JA$\rightarrow$EN) using beam search with a beam size 5 and length penalty 1.0.

Decoding At inference time, the most straightforward way is to generate the sequence with the highest probability at each position. The outputs from the CTC-based NAT models require an additional collapse process Γ^{-1} which can be done instantly. A relatively more accurate method is to decode multiple sequences, and rescore them to obtain the best candidate in parallel, i.e. *noisy parallel decoding* [NPD, 55]. Furthermore, CTC-based models are also capable of decoding sequences using beam-search [110], and optionally combined with n-gram language models [63, 74]. More precisely, we search in a beam to approximately find the optimal $\boldsymbol{y}^*$ that maximizes:

$$\log p_\theta(\boldsymbol{y}|\boldsymbol{x}) + \alpha \cdot \log p_{\text{LM}}(\boldsymbol{y}) + \beta \log |\boldsymbol{y}| \tag{6.6}$$

where α and β are hyperparameters for language model scores and word insertion bonus. In principle, it is no longer non-autoregressive as beam-search is a sequential process by nature. However, it does not contain any neural network computations and can be implemented efficiently in C++ [2].

Baselines We adopt transformer (AT) and existing NAT approaches (see Table 6.2) for comparison. For AT, except for the standard *base* and *big* architectures [178], we also compare with a

[1] `https://github.com/bitextor/bicleaner`
[2] `https://github.com/parlance/ctcdecode`

Models		Iter.	Speed	WMT'14		WMT'16	
				EN-DE	DE-EN	EN-RO	RO-EN
AT	Transformer *base* (teacher)	N	1.0×	**27.48**	**31.39**	**33.70**	**34.05**
	Transformer *base* (12-1)	N	2.4×	26.21	30.80	33.17	33.21
	+ KD	N	2.5×	27.34	30.95	33.52	34.01
Iterative NAT	iNAT [102]	10	1.5×	21.61	25.48	29.32	30.19
	Blockwise [163]	$\approx N/5$	3.0×	27.40	-	-	-
	InsT [164]	$\approx \log N$	4.8×	27.41	-	-	
	CMLM [48]*	10	1.7×	27.03	30.53	33.08	33.31
	LevT [56]	Adv.	4.0×	27.27	-	-	33.26
	KERMIT [15]	$\approx \log N$	-	27.80	30.70	-	-
	LaNMT [160]	4	5.7×	26.30	-	-	29.10
	SMART [50]*	10	1.7×	27.65	31.27	-	-
	DisCO [72]*	Adv.	3.5×	27.34	31.31	33.22	33.25
	Imputer [150]*	8	3.9×	**28.20**	**31.80**	**34.40**	**34.10**
Fully NAT	Vanilla-NAT [55]	1	15.6×	17.69	21.47	27.29	29.06
	LT [70]	1	3.4×	19.80	-	-	-
	CTC [110]	1	-	16.56	18.64	19.54	24.67
	NAT-REG [185]	1	-	20.65	24.77	-	-
	Bag-of-ngrams [156]	1	10.0×	20.90	24.60	28.30	29.30
	Hint-NAT [109]	1	-	21.11	25.24	-	-
	DCRF [167]	1	10.4×	23.44	27.22	-	-
	Flowseq [117]	1	1.1 ×	23.72	28.39	29.73	30.72
	ReorderNAT [146]	1	16.1×	22.79	27.28	29.30	29.50
	AXE [49]*	1	15.3×	23.53	27.90	30.75	31.54
	EM+ODD [166]	1	16.4×	24.54	27.93	-	-
	GLAT [138]	1	15.3×	25.21	29.84	31.19	32.04
	Imputer [150]*	1	18.6×	25.80	28.40	32.30	31.70
	Ours (Fully NAT)	1	17.6×	11.40	16.47	24.52	24.79
	+ KD	1	17.6×	19.50	24.95	29.91	30.25
	+ KD + CTC	1	16.8×	26.51	30.46	33.41	34.07
	+ KD + CTC + VAE	1	16.5×	**27.49**	31.10	**33.79**	33.87
	+ KD + CTC + GLAT	1	16.8×	27.20	**31.39**	33.71	**34.16**

Table 6.2: Comparison between our models and existing methods. The speed-up is measured on WMT'14 En→De test set. **Iter.** denotes the number of iterations at inference time, **Adv.** means adaptive, * denotes models trained with distillation from a *big* Transformer.

Models	**WMT'14**		**WMT'16**	
	EN-DE	**DE-EN**	**EN-RO**	**RO-EN**
Transformer *base* (teacher)	**46.2**	34.9	**46.4**	39.9
Transformer *base* (12-1)	44.7	33.4	45.3	38.4
+ KD	45.6	34.6	45.8	38.6
Ours (Fully NAT)				
KD + CTC + VAE	45.9	35.0	46.1	**42.2**
KD + CTC + GLAT	45.8	**35.3**	45.9	40.6

Table 6.3: Comparison between our models and autoregressive models in terms of METEOR [32].

deep encoder, shallow decoder transformer suggested in Kasai et al. [73] that follows the model dimensions of *base* with 12 encoder layers and 1 decoder layer (i.e. *base* (12-1) for short).

Evaluation BLEU [133] is used to evaluate the translation performance for all models. Following prior works, we compute tokenized BLEUs for EN↔DE and EN↔RO, while using SacreBLEU [137] for JA→EN. We also test our models by METEOR [32][3]. In this chapter, following Kasai et al. [73], we use three measures to fully investigate the translation latency of all the models:

- $\mathcal{L}_1^{\text{GPU}}$: translation latency by running the model with one sentence/batch on single GPU, aligning applications like instantaneous translation.
- $\mathcal{L}_1^{\text{CPU}}$: the same as $\mathcal{L}_1^{\text{GPU}}$ while running the model without GPU speed-up. Compared to $\mathcal{L}_1^{\text{GPU}}$, it is less friendly to NAT models that make use of parallelism, however, closer to real scenarios.
- $\mathcal{L}_{\text{max}}^{\text{GPU}}$: the same as $\mathcal{L}_1^{\text{GPU}}$ on GPU while running the model in a batch with as many sentences as possible. In this case, the hardware memory bandwidth is taken into account.

We measure the wall-clock time for translating the whole test set, and report the averaged time over sentences as the latency measure. For more implementation details, please refer to Appendix 6.6.

Results

WMT'14 EN↔DE & WMT'16 EN↔RO We report the performance of our fully NAT model comparing with AT and existing NAT approaches (including both iterative and fully NAT models) in Table 6.2. Iterative NAT models with enough number of iterations generally outperform fully NAT baselines by a certain margin as they are able to recover the generation errors by explicitly

[3]`https://www.cs.cmu.edu/~alavie/METEOR/README.html`

	Configuration	BLEU (Δ)	BP	$\mathcal{L}_1^{\text{GPU}}$ (Speed-up)		$\mathcal{L}_1^{\text{CPU}}$ (Speed-up)	
AT	*big* (teacher)	21.07	0.920	345 ms	1.0 ×	923 ms	1.0 ×
	base	18.91	0.908	342 ms	1.0 ×	653 ms	1.4 ×
	base (12-1)	15.47	0.806	152 ms	2.3 ×	226 ms	4.0 ×
	base (12-1) + KD	18.76	0.887	145 ms	2.4 ×	254 ms	3.6 ×
NAT	KD + CTC	16.93 (+0.00)	0.828	17.3 ms	19.9 ×	84 ms	11.0 ×
	KD + CTC + VAE	18.73 (+1.80)	0.862	16.4 ms	21.0 ×	83 ms	11.1 ×
	w. *BS20*	19.80 (+2.87)	0.958	28.5 ms	12.1 ×	99 ms	9.3 ×
	w. *BS20* + *LM*	**21.41 (+4.48)**	0.954	31.5 ms	11.0 ×	106 ms	8.7 ×
	w. *NPD5*	18.88 (+1.95)	0.866	34.9 ms	9.9 ×	313 ms	2.9 ×
	w. *NPD5* + *BS20* + *LM*	**21.84 (+4.91)**	0.962	57.6 ms	6.0 ×	284 ms	3.2 ×

Table 6.4: Performance comparison between fully NAT and AT models on WMT'20 JA→EN. Translation latency on both the GPU and CPUs are reported over the test set. The brevity penalty (BP) is also shown for reference.

modeling dependencies between (partially) generated tokens. However, the speed advantage is relatively small compared to AT *base* (12-1) which also achieves 2.5 times faster than the AT baseline.

Conversely, our fully NAT models are able to readily achieve over 16 times speed-up on EN→DE by restricting translation within a single iteration. Surprisingly, merely training NAT with KD and CTC loss already beats the state-of-the-art for single iteration NAT models across all four directions. Moreover, combining with either latent variables (VAE) or glancing targets (GLAT) further closes the performance gap. No significant difference has been found between transformer-base and our best models via `compare-mt` [128]. We also employ METEOR [32] to evaluate our best models and results are shown in Table 6.3. We see that our model can achieve similar performance compared to the transformer-base model.

Table 6.2 also indicates the difficulties of learning NAT on each dataset. For instance, EN↔RO is relatively easier as "KD + CTC" is enough to close the performance gap. By contrast, applying VAE or GLAT helps to capture non-monotonic dependencies and improve by $0.5 \sim 1$ BLEU points on EN↔ED. For both datasets, we ONLY need a single greedy generation to achieve similar translation quality as AT beam-search results (beam size 4).

WMT'20 JA→EN In Table 6.4, we also present results for training the fully NAT model on a more challenging benchmark – WMT'20 JA→EN which is much larger (13M pairs) and noisier. In addition, JA is linguistically distinct from EN which makes it harder to learn mappings

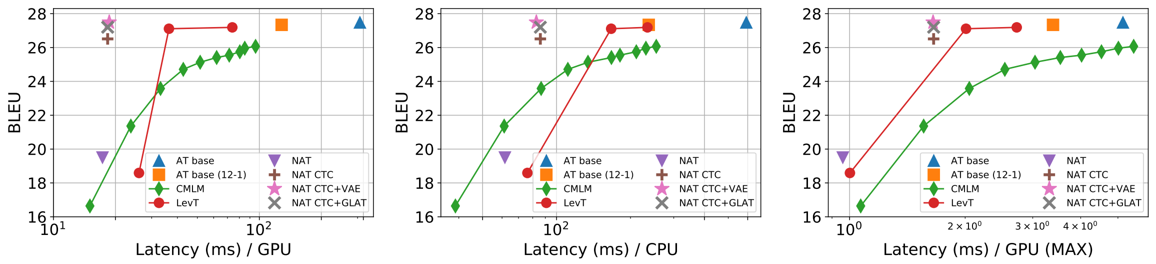


Figure 6.3: Quality v.s. Latency (the upper left corner achieves the best trade-off) for fully NAT models with other translation models (AT *base* and *base* 12-1 [73], CMLM [48] and LevT [56]) on WMT'14 EN→DE. We evaluate latency in three setups (from left to right: $\mathcal{L}_1^{\text{GPU}}$, $\mathcal{L}_1^{\text{CPU}}$, $\mathcal{L}_{\text{max}}^{\text{GPU}}$) and show them in Logarithmic scale for better visualization.

between them. Consequently, both AT (12-1) and our fully NAT models become less confident and tend to generate shorter translations (BP < 0.9), which in turn underperform the AT teacher even trained with KD.

Beam search & NPD The proposed NAT can be further boosted by allowing beam-search or re-ranking (NPD) after prediction. For CTC beam search, we use a fixed beam-size 20 while grid-search α, β (Equation(6.6)) based on the performance on the validation set. The language model [4] is trained directly on the distilled target sentences to avoid introducing additional information. For noisy parallel decoding (NPD), we draw multiple $\boldsymbol{z}$ from the learned prior distribution with temperature 0.1, and use the teacher model to rerank the best $\boldsymbol{z}$ with the corresponded translation.

As shown in Table 6.4, with similar GPU latency ($\mathcal{L}_1^{\text{GPU}}$), beam search is much more effective than NPD with re-ranking, especially combined with a 4-gram LM where we achieve a BLEU score of 21.41, beating the teacher model with $11\times$ speed-up. More importantly, by contributing the insertion bonus (3rd term in Equation 6.6) with β in beam search, we have the explicit control to improve BP and output longer translations. Also, we gain another half point by combining NPD and beam search. To have a fair comparison, we also report latency on CPUs where it is limited to leverage parallelism of the device. The speed advantage drops rapidly for NAT models, especially for NAT with NPD, however, we still maintain around 100 ms latency via beam search – over $2\times$ faster than the lightweight AT (12-1) systems with higher translation quality.

Quality v.s. Latency We perform a full investigation for the trade-off between translation quality and latency under three measures defined in Section 6.4. The results are plotted in Figure 6.3. For fully NAT models, no beam search or NPD is considered. In all three setups, our fully NAT models obtain superior trade-off compared with AT and iterative NAT models. Itera-

[4]`https://github.com/kpu/kenlm`

tive NAT models (LevT and CMLM) require multiple iterations to achieve reliable performance with the sacrifice of latency, especially under $\mathcal{L}_1^{\text{CPU}}$ and $\mathcal{L}_{\max}^{\text{GPU}}$ where iterative NAT performs similarly or even worse than AT *base* (12-1), leaving fully NAT models a more suitable position in quality-latency trade-off.

Figure 6.3 also shows the speed advantage of fully NAT models shrinks in the setup of $\mathcal{L}_1^{\text{CPU}}$ and $\mathcal{L}_{\max}^{\text{GPU}}$ where parallelism is constrained. Moreover, NAT models particularly those with CTC consume more computation and memory compared to AT models with a shallow decoder. For instance when calculating $\mathcal{L}_{\max}^{\text{GPU}}$, we notice that the maximum allowed batch is 120K tokens for AT base (12-1), while we can only compute 15K tokens at a time for NAT with CTC due to the up-sampling step, even though the NAT models still win the wall-clock time. We mark it as one limitation for future research.

Ablation Study

Impact of various techniques Our fully NAT models benefit from dependency reduction techniques in four aspects (data, model, loss function and learning), and we analyze their effects on translation accuracy through various combinations in Table 6.5. First of all, the combinations without KD have a clear performance drop compared to those with KD, showing its vital importance in NAT training. For the loss function, although both AXE [48] and CTC consider the latent alignments, the CTC-based model obtains much better accuracy due to its flexibility of output length. In all cases, incorporating latent variables also effectively improves the accuracy, especially for CTC without KD (~ 5 BLEU improvement). Because of the capability to reduce the mismatch between training and inference time, the model with GLAT is superior to those with randomly (RND) sampled masks. To conclude, we find that KD and CTC are necessary components for a robust fully NAT model. Adding either VAE or GLAT to them achieves similar improvements.

Distillation corpus We report the performance of models trained on real data and distilled data generated from AT *base* and *big* models in Table 6.6. For *base* models, both AT (12-1) and NAT achieve better accuracy with distillation, while AT benefits more by moving from *base* to *big* distilled data. On the contrary, the NAT model improves marginally indicating that in terms of the modeling capacity, our fully NAT model is still worse than AT model even with 1 decoder layer. It is not possible to further boost the NAT performance by simply switching the target to a better distillation corpus, which aligns with the finding in Zhou et al. [202]. Nonetheless, we can increase the NAT capacity by learning in *big* size. As shown in Table 6.6, we can achieve superior accuracy compared to AT (12-1) with little effect on the translation latency ($\mathcal{L}_1^{\text{GPU}}$).

KD	AXE	CTC	VAE	RND	GLAT	BLEU
						11.40
✓						19.50
	✓					16.59
✓	✓					21.66
		✓				18.18
✓		✓				**26.51**
		✓	✓			23.58
✓	✓		✓			22.19
✓		✓	✓			**27.49**
✓	✓			✓		22.74
✓	✓				✓	24.67
✓		✓		✓		26.16
		✓			✓	21.81
✓		✓			✓	**27.20**
✓		✓	✓		✓	**27.21**

Table 6.5: Ablation on WMT'14 EN→DE test set with different combinations of techniques. The default setup shows a plain NAT model [55] directly trained on raw targets with the cross entropy (CE) loss.

Effective Latent Dimensionality of Latent Variables To confirm the necessity of combining VAEs with CTC, We apply principal component analysis (PCA) [188] on the learned latent variables. More precisely, we extract the latent variables from the posterior of various models (see Table 6.5) on WMT'14 EN→DE test set. These main components' explained variance ratios, the percentage of variance that is attributed by each of the components, are shown in Figure 6.4.

First, we find that the number of effective latent dimensionality (capturing at least 95% of the total variance) is much lower than the number of latent dimensions (8 in our experiments), which indicates simply increasing the number of latent dimensions does not lead to better representations, and the ability to capture dependencies is limited. Therefore, VAEs need to be combined with other techniques e.g. KD, CTC to take effect. Also, compared to the AXE, the effective dimensionality of latent variables in CTC loss-based models is higher. We include more analysis with qualitative examples in Appendix 6.6.

Models		Distillation *base*	Distillation *big*	BLEU	Speed-up
AT	*base*			27.43	1.0×
	big			28.14	0.9×
	base (12-1)			26.12	2.4×
		✓		27.34	2.5×
			✓	27.83	2.4×
NAT	*base*			23.58	16.5×
		✓		27.49	16.5×
			✓	27.56	16.5×
	big		✓	**27.89**	15.8×

Table 6.6: Performance comparison between AT and NAT models on the test set of WMT'14 EN→DE. The latency is measured one sentence per batch and compared with the Transformer *base*. For NAT model, we adopt CTC+VAE as the basic configuration.

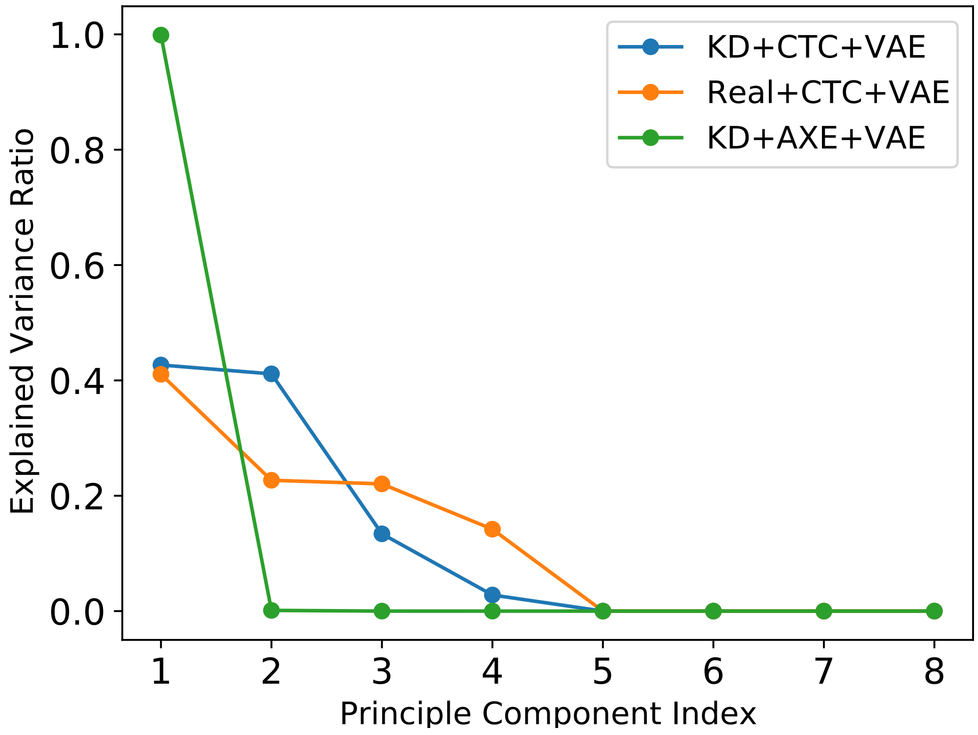


Figure 6.4: Principle component explained variance ratios of latent variables on WMT'14 EN→DE test set.

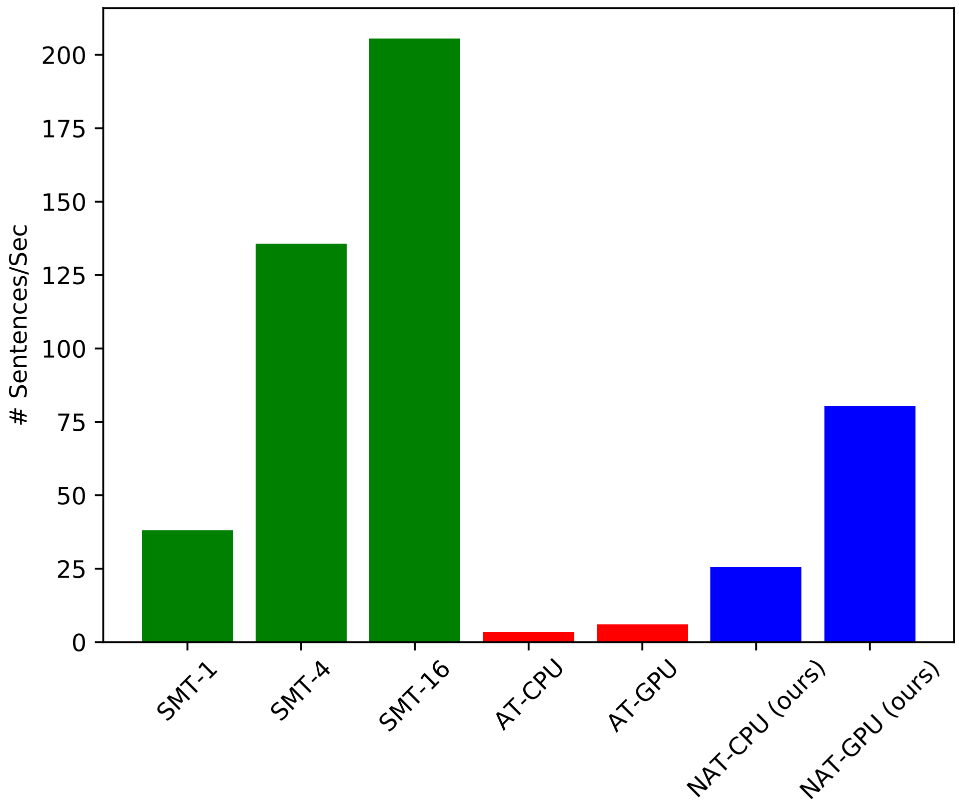


Figure 6.5: Decoding Speed Comparison of various machine translation models. SMT-16 denotes phrase-based statistical machine translation decoding with 16 CPU threads.

Comparison with Non-Neural MT Here we compare the decoding speed of various machine translation systems. We train a standard phrase-based statistic machine translation [91] by NiuTrans [192]. We test all models on the WMT'14 DE-EN testset with batch size 1 and the result is shown in Figure 6.5. NAT models decoding on GPU can surpass the SMT with a single CPU thread but is still much slower as compared to SMT multi-threads decoding. How to close this decoding speed gap is worthwhile to be explored in the future. Some widely-adopted techniques such as knowledge distillation [77] and model pruning may be helpful.

6.5 Summary

In this chapter, we aim to minimize the performance gap between fully NAT and AT models. We investigate *dependency reduction* methods from four perspectives and carefully unite them with necessary revisions. Experiments on three translation benchmarks demonstrate that the proposed fully NAT models achieve the SoTA performance. For future work, it is worth exploring simpler but more effective diagrams for learning NAT models. For instance, with the combination of CTC and more powerful latent variable models, it is possible to remove the necessity of knowledge distillation. In the future, one interesting direction to apply these techniques to pre-train a non-autoregressive LM on a large-scale data.

6.6 Appendix

Implementation Details

Architecture We design our fully NAT model with the hyperparameters of the *base* Transformer: 8-512-2048 [178]. For EN$\rightarrow$DE experiments, we also implement the NAT model in *big* size: 8-1024-4096 for comparison.

VAEs For experiments using variational autoencoders (VAE), we use the last layer encoder hidden states to predict the mean and variance of the prior distribution. The latent dimension D is set to 8, and the predicted $\boldsymbol{z}$ are linearly projected and added on the encoder outputs. Following Shu et al. [160], we use a 3 layer encoder-decoder as the posterior network, and apply freebits annealing [18] to avoid posterior collapse.

CTC By default, we upsample the length of decoder inputs $3\times$ as long as the source for CTC, while using the golden length for other objectives (CE and AXE). We also train an additional length predictor when CTC is not used. For both cases, we use *SoftCopy* [186] which interpolated the encoder outputs as the decoder inputs based on the relative distance of source and target positions.

GLAT The mask ratio, f_{ratio}, is 0.5 for GLAT training. The original GLAT [138] assumes to work with the golden length so that it can glance at the target by placing the target word embedding to a clear corresponded inputs. It is incompatible with CTC loss where we always need longer inputs than the targets. To enable GLAT learning, we glance at target tokens from the viterbi aligned tokens ($\boldsymbol{\alpha} = \arg\max_{\boldsymbol{\alpha}\in\beta(\boldsymbol{y})} p(\boldsymbol{\alpha}|\boldsymbol{x})$) which has the same length as the decoder inputs.

Training For both AT and NAT models, we set the dropout rate as 0.3 for EN$\leftrightarrow$DE and EN$\leftrightarrow$RO, and 0.1 for JA$\rightarrow$EN. We apply weight decay 0.01 as well as label smoothing $\epsilon = 0.01$. All models are trained for 300K updates using Nvidia V100 GPUs with a batch size of approximately 128K tokens. We measure the validation BLEU scores for every 1000 updates, and average the last 5 checkpoints to obtain the final model.

Inference We measure the GPU latency by running the model on a single Nvidia V100 GPU, and CPU latency on Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz with 80 cores. All models are implemented on `fairseq` [132].

More ablation study

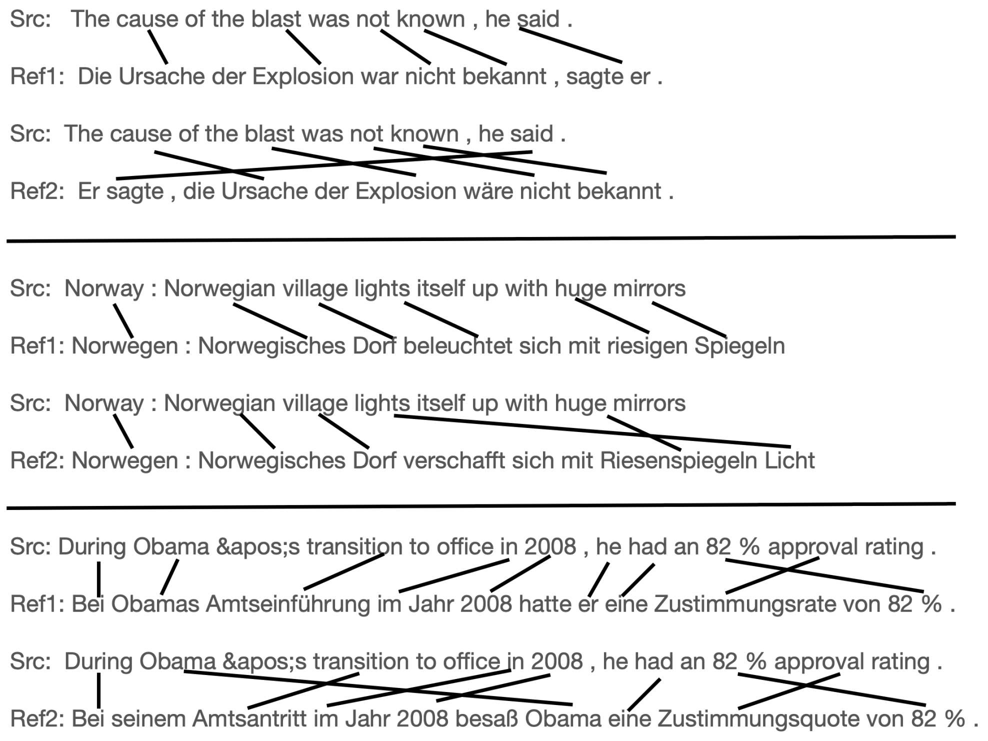
λ	BLEU	$\mathcal{L}_1^{\text{GPU}}$	$\mathcal{L}_{\max}^{\text{GPU}}$	$\mathcal{L}_1^{\text{CPU}}$
1.5	26.16	17.9 ms	**0.95 ms**	**66.6 ms**
2.0	26.39	17.5 ms	1.03 ms	71.6 ms
2.5	**26.54**	17.6 ms	1.16 ms	76.9 ms
3.0	26.51	**17.0 ms**	1.32 ms	81.8 ms

Table 6.7: Performance comparison of different upsample ratios (λ) for CTC-based models on WMT'14 EN$\rightarrow$DE test set. All models are trained on distilled data.

Upsampling Ratio (λ) for CTC Loss To meet the length requirements in CTC loss, we upsample the encoder output by a factor of 3 in our experiments. We also explore other possible values and report the performance in Table 6.7. The higher upsampling ratio provides a larger alignment space, leading to better accuracy. Nevertheless, with a large enough sampling ratio, a further increase will not lead to the performance increase. Because of the high degree of parallelism, $\mathcal{L}_1^{\text{GPU}}$ speed is similar among these ratios. However, the model with a larger ratio has a clear latency drop on CPU or GPU with large batches.

Representation reordering in the latent space In our main experiments, VAEs has been proven to effectively improve the performance of NAT models. Here, we perform a qualitative study to show how VAEs helps NAT models.

Ott et al. [130] collected additional reference translations for each source sentence in the WMT'14 En$\rightarrow$De test set. We first choose three source sentences and show the alignments between them and two of their different translations in Figure 6.6. In each sample, it is clear to find that the word order of the first pair is more similar to the second one (e.g., in the second sample, the verb 'light' in the source sentence is translated to the end of the second reference sentence). However, given the monotonic alignment assumption, CTC is difficult to align sentence pairs with different word orders. Then, for each sample, we extract latent variables of both sentence pairs and align them by first computing the Euclidean distance between every position and then employing the linear sum assignment algorithm (LAP).



Src: The cause of the blast was not known , he said .
Ref1: Die Ursache der Explosion war nicht bekannt , sagte er .
Src: The cause of the blast was not known , he said .
Ref2: Er sagte , die Ursache der Explosion wäre nicht bekannt .
Src: Norway : Norwegian village lights itself up with huge mirrors
Ref1: Norwegen : Norwegisches Dorf beleuchtet sich mit riesigen Spiegeln
Src: Norway : Norwegian village lights itself up with huge mirrors
Ref2: Norwegen : Norwegisches Dorf verschafft sich mit Riesenspiegeln Licht
Src: During Obama &apos;s transition to office in 2008 , he had an 82 % approval rating .
Ref1: Bei Obamas Amtseinführung im Jahr 2008 hatte er eine Zustimmungsrate von 82 % .
Src: During Obama &apos;s transition to office in 2008 , he had an 82 % approval rating .
Ref2: Bei seinem Amtsantritt im Jahr 2008 besaß Obama eine Zustimmungsquote von 82 % .

Figure 6.6: Alignments between source sentences and their different translations.

Regarding the first pair as the baseline, we find that the latent variable is able to adjust the word order according to the input sentence pair. For example, the alignment between latent variables of the second sample is shown as: 0-0, 1-1, 2-2, 3-3, 4-9, 5-5, 6-6,7-7, 8-8, 9-4, which shows that the latent representation of the 9th position in the second pair is aligned to the 5th position of the second pair. In another word, the latent representation of the word ’lights’ is reordered to the last position in the second pair’s latent variable, which corresponds to the word order difference in the second pair. Therefore, given various reference information, the latent variable makes the alignment between the source and target representation more monotonic. CTC can consequently benefit from it to learn a better NAT model.

Part III

Capacity Allocation in Multilingual Neural Machine Translation

Chapter 7

Multilingual Neural Machine Translation with Deep Encoder and Multiple Shallow Decoders

In the decoding process of the standard transformer-based NMT model, the representation of the source sequence is obtained by the bidirectional transformer encoder with high parallelism. However, at the decoder side, because translations are predicted one-by-one conditioned on all previous words, the decoder will be called many times in a sequential order. Consequently, the latency is highly correlated with the computational complexity of the decoder. Besides improving the parallelizability of the decoder, recently, Kasai et al. [73] show that given a fixed capacity budget, as measured by the number of layers, models with a deep encoder and a shallow decoder (DESD) are faster at inference time when compared to standard models while maintaining translation quality.

From this chapter, we extend this capacity allocation idea to multilingual neural machine translation (MNMT) which is capable of handling multiple language pairs in a single model. In this chapter, we first explore the speed-accuracy trade-off of MNMT through model capacity allocations. According to our observations, we propose a deep encoder, (multiple) shallow decoder(s) architecture to achieve a superior speed-accuracy trade-off for one-to-many and many-to-one translation tasks.

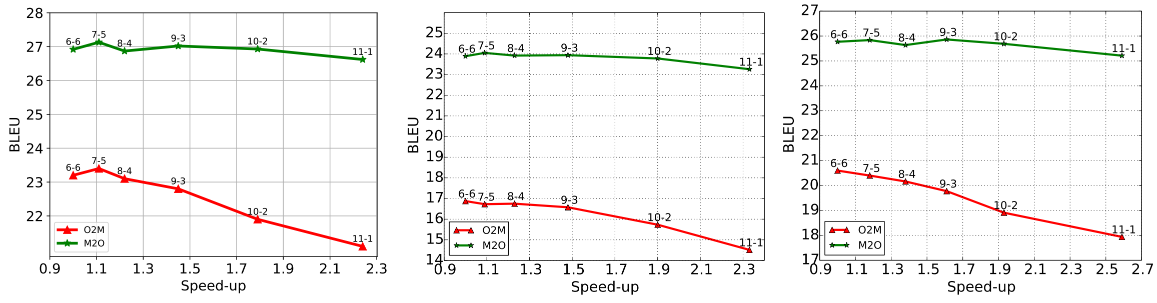
7.1 Introduction

Encoder-decoder based neural machine translation (NMT) systems have achieved great success on bilingual translation tasks [6, 22, 45, 168, 178]. Recently, multilingual neural machine translation (MNMT) has also attracted much attention because of its ease of deployment, knowledge transfer among languages and the potential to enable zero-shot translation [4, 34, 41, 58, 68, 199]. While MNMT can support translations in several directions, not all of them have better performance when compared to their corresponding bilingual models. Suspecting that poor performance in some directions is due to the limited model capacity, many prior works adopt deeper encoder and decoder [182, 198, 199]. However, increasing the number of layers, especially in the decoder, deteriorates the latency of translation and memory costs. Recently, Kasai et al. [73] show that given a fixed capacity budget, as measured by the number of layers, models with a deep encoder, shallow decoder (DESD) architecture are faster at inference time when compared to standard models with an equal number of encoder and decoder layers while maintaining translation quality.

Inspired by findings from Kasai et al. [73], in this chapter, we explore the speed-accuracy trade-off in multilingual machine translation systems. Given the same model capacity budget, we experiment various layer allocation strategies and analyze multilingual models in the one-to-many (O2M) and many-to-one (M2O) settings. In the one-to-many setting, there are numerous target languages from a single source language (limited to English in this study); and in the many-to-one setting, several possible source languages are translated into a single target language (again, English in this study).

In the many-to-one scenario, we find that allocating more capacity to the encoder reduces the latency while achieving comparable performance. We hypothesize that a deeper encoder helps the model accommodate multiple source languages, while a shallow decoder is sufficient to support a single target language [73].

However, in the one-to-many translation setting, speed-accuracy trade-off is complicated. We observe a performance drop as the decoder depth is reduced. We hypothesize that the shallow decoder can no longer model several different target languages adequately. With the goal of obtaining low latency while maintaining translation quality, we propose using *multiple* shallow decoders where each decoder is responsible for a subset of the target languages. Clearly, the introduction of multiple shallow decoders increases the size of our model. However, at inference time, given a specific target language, only one shallow decoder will be used, thus not adding latency or memory costs. With multiple target languages and decoders, one natural question is how to assign each target language to one of these decoders. We investigate several methods to assign each target language to one of these shallow decoders. More details are in the Section 7.3. Experimental results on three multilingual translation corpora show the effectiveness of our method to improve translation accuracy with lower latency at the same time.



(a) OPUS-100 (b) TED8-Related (c) TED8-Diverse

Figure 7.1: Speed and accuracy trade-off of various layer allocations for O2M and M2O translations on OPUS-100, TED8-Related and TED8-Diverse corpora. X-Y denotes X and Y layers in the encoder and decoder respectively. Best viewed in color.

7.2 Deep encoder and shallow decoder (DESD) for multilingual NMT

Background The transformer-based NMT model [178] achieves state-of-the-art performance on many translation tasks. It consists of an encoder and a decoder, each of which contains several stacked layers. Since the transformer relies entirely on the attention mechanism, it allows more parallelization compared to recurrent neural networks. Specifically, at training time, the computation can be parallelized both in the encoder and decoder. At inference time, due to the auto-regressive property, the decoder needs to generate tokens one by one. However, the computation in the encoder is still parallelized given the source sentence. Therefore, the main latency of the transformer at inference time happens in the decoder, especially translating long sentences. Recently, Kasai et al. [73] find that on bilingual machine translation tasks, putting more capacity of the transformer model to the encoder substantially reduces the decoding time and maintains the performance at the same time.

Because this deep encoder and shallow decoder model achieves a superior speed-accuracy trade-off on bilingual translation tasks, in this section, we try to understand the layer allocations of transformer on the multilingual neural machine translation task given the same capacity budget which is measured by the number of layers in the encoder and decoder. We first experiment with three multilingual translation corpora.

- OPUS-100 is a massively multilingual dataset collected from OPUS [174], including 100 languages in total with 99 languages to-and-from English. It consists of 55 million training sentence pairs with up to 1M samples per language pair, and covers 94 dev/test language pairs, each with 2000 samples at most.

- TED8-Related [184]: 4 low resource languages (Azerbaijani: az, Belarusian: be, Glacian: gl, Slovak: sk) and 4 relevant high resource languages (Turkish: tr, Russian: ru, Portuguese: pt, Czech: cs)
- TED8-Diverse [184]: 8 languages without consideration for relatedness (Bosnian: bs, Marathi: hr, Hindi: hi, Macedonian: mk, Greek: el, Bulgarian: bg, French: fr, Korean: ko)

Instead of just trying the shallowest possible decoder (1-layer), we train models with various configurations on these three corpora. Other than the layer allocation, all the other hyper-parameters and model configurations are the same among these models and the same training procedure is applied to these models (model and training details are listed in Appendix 7.7.). To understand the speed and accuracy trade-off of the layer allocation, two metrics are reported:

- BLEU: the average BLEU score [133] over all directions.
- DS: the decoding speed. It is measured by the number of tokens per second the system translates given one sentence at a time on a single GPU.

The results are shown in Figure 7.1. Generally, we see that models with fewer decoder layers obtain higher decoding speed.

Many-to-one (M2O) translation

In the M2O translation, there is no significant performance difference among these layer allocations. We hypothesize that this is because the deeper encoder learns better representations from a large number of source languages while on the decoder side only one language needs to be modeled. Therefore, given a more robust representation of source languages, the shallow decoder is able to generate high-quality translations. For example, the model with 10 encoder layers and 2 decoder layers obtains slightly better performance and a 1.8$\times$ speedup at the same time.

One-to-many (O2M) translation

However, in the O2M translation setting, although models with the shallower decoder have lower latency compared to the standard transformer (6-6), there is a clear performance drop in terms of translation accuracy, especially for models with just 1 or 2 decoder layers. We attribute this to the shallow decoder not having enough capacity to model a large number of target languages.

7.3 Deep Encoder and Multiple Shallow Decoders (DEMSD)

We have seen that in one-to-many translation, DESD models have a performance drop compared to the standard transformer. In order to preserve translation quality and low latency at the same time, we propose a model with a shared deep encoder and *multiple* shallow decoders (DEMSD), each of which is used to decode a subset of target languages. Although this will introduce more parameters, at inference time only one shallow decoder is needed for a given translation (since the output language is fixed) thus the model incurs no extra latency or memory costs. One natural question that arises when using this multiple-decoder approach is how to assign output languages to each of the decoders. In this section, we explore several language assignment methods to assign each target language (or language group) to one of these multiple decoders. As a result, each decoder only needs to handle a disjoint subset of target languages.

One language per decoder (EACH)

The simplest way is to use a separate decoder for each output language. As a result, we will have as many decoders as the number of target languages and each decoder only needs to model one language.

Random language set per decoder (RAND)

In this method, we assign a random set of languages to a single decoder. As the performance of the model will vary significantly based on the random assignment, we repeat this scheme with three different random assignments and report the average results. Instead of completely random grouping languages, we let each decoder handle the same number of languages but languages in one decoder are randomly grouped.

One language family per decoder (FAM)

Another intuitive way for language assignment is to use linguistic features [26, 36, 106], such as language family, typology, etc. In this method, we are guided by the intuition that languages from the same linguistic family share similar features which might be captured by a single decoder resulting in better performance. Thus, we group languages into several sets based on their linguistic families, and assign a family of languages to each decoder. As a result, we will have as many decoders as the number of language families in the target languages. We expect that in the same decoder, a better knowledge transfer will happen among languages in the same language family. For example, in the TED8-Related corpus, 8 target languages are split into 4 languages families which are TURKIC, SLAVIC, ROMANCE and CZECH–SLOVAK. The details are shown

Language Family	Languages
TURKIC	az, tr
SLAVIC	be, ru
ROMANCE	gl, pt
CZECH–SLOVAK	sk, cs

Table 7.1: Language families in the TED8-Related corpus.

in Table 7.1. The language family-based assignment results on other corpora are shown in Appendix 7.7.

Pre-trained language embedding based assignment (EMB)

From Johnson et al. [68], a common way to indicate the target language is prepending a target language token to the source sentence. With the goal of capturing the information of languages they represent, their embeddings are trained end-to-end with source-target sentence pairs. We call these embeddings as the language embeddings here. According to Johnson et al. [68], these language embeddings are able to capture target language features. Therefore, we first extract them from a well-trained model and group target languages according to them. Finally, each group is assigned to one of these decoders.

Self-taught assignment (ST)

One disadvantage of the pre-trained language embedding based grouping method is the need of a pre-trained machine translation model. It would be better if the model assigns each target language to one of these multiple shallow decoders during the training automatically. We expect that given a fixed number of decoders and target languages, the model is capable of choosing the most appropriate decoders for each language.

Specifically, our model consists of a shared encoder, E, and N multiple decoders, $D = [D_1, D_2, ..., D_N]$. Given a language L, the model will choose a decoder, D_i for training and translation so that the log probability of output sequence y given the input sequence x is $\log p(y|x, E, D_i)$ where i=$\arg\max_j p(j|L_e)$ and $p(\cdot|L_e)$ is the probability of each decoder being chosen given the language L and its language embedding vector L_e. Intuitively, our model will learn the distribution of each decoder being chosen given a language and choose the one with the highest probability. However, the $\arg\max$ operation here is non-differentiable thus during training we consider the Gumbel-Softmax [67], a differentiable approximation of the $\arg\max$ operation.

In Gumbel-Softmax, it models the $p(j|L_e)$ as:

$$p(j|L_e) = \frac{\exp(l_j + g_j)/\tau}{\sum_{k=1}^{N} \exp(l_k + g_k)/\tau} \tag{7.1}$$

where l is the logit and g=$-\log(-\log(u))$ and $u \sim \mathcal{U}(0,1)$. In the forward pass, the differentiable approximation of the $\arg\max$ operation is used to choose the decoder for the input language and during the backward, the true gradient of the Straight-Through Gumbel-Softmax outputs is used. In our experiments, the temperature τ is linearly reduced from 5 to 0.5. Finally, during training, the probability of the target sequence y given the source sentence x and multiple decoders D is:

$$p(y|x) = \sum_{n=1}^{N} p(n|L_e)p_n(y|x) \tag{7.2}$$

where $p_n(y|x)$ is the probability of y given x in the n-th decoder and $p(n|L_e)$ is the probability of the n-th decoder being sampled given the embedding of the language L. During inference, only the decoder with the highest probability will be used to decode the input sentences.

7.4 Experiments

Experimental setup

We conduct experiments on TED8-Related, TED8-Diverse and OPUS-100 multilingual machine translation corpora. We mainly follow the data preprocessing settings in previous works [184, 199].

Hyperparameters On OPUS-100, we follow most of the standard hyperparameters in the transformer-base [178]: 8 attention heads per layer, 512 model dimensions, 2048 hidden dimensions and 0.1 dropout. We train batches of 128k tokens using Adam [81] with β = (0.9, 0.98) and ϵ = 10^{-6} and 0.1 label smoothing. Following [199], the learning rate goes to $4\text{e}{-4}$ within 4,000 steps, and then decays with the inverse square-root schedule. All models are trained with 300,000 steps. Furthermore, to mitigate the training data imbalance issue, the temperature sampling method is adopted [4] which is set as 5 in all experiments.

On TED8 corpora, following Wang et al. [184], a smaller transformer model with 512 model dimensions, 1024 hidden dimensions and 0.3 dropout is adopted. All models are trained for 40k steps with batches of 16k tokens and a smaller learning rate $2\text{e}{-4}$. The other training procedure is the same as the OPUS-100.

Evaluation metrics For all models, we evaluate on the checkpoint with the best validation loss and use beam size 4 and length penalty 0.6 in decoding. Besides reporting the average translation accuracy over all languages, on OPUS-100, we predefine high ($\geq$ 1M) and low ($<$ 1M) resource languages according to their training data sizes and average scores on each of them are also computed. To assess the translation accuracy, we employ BLEU [133], BERTScore [201] and COMET (wmt20-comet-da) [147] scores. Furthermore, we employ `comet-compare` to compare various models to get statistical significance [88] when necessary. For the evaluation speed, **DS**, it is measured by the number of tokens the system translates per second given one sentence at a time on a single GPU.

ID	Model	TP	DP	Speed	All BLEU	All COMET	All BS	Low BLEU	Low COMET	Low BS	High BLEU	High COMET	High BS
1	6-6	77	77	1.0×	23.2	0.186	83.1	25.7	0.265	84.6	21.1	0.117	81.8
2	11-1	72	72	2.2×	21.1	0.060	82.5	23.2	0.168	84.0	19.3	-0.033	81.1
3	11-1-EACH	489	72	2.2×	21.2	0.077	82.4	21.7	0.081	83.2	20.8	0.074	81.7
4	11-1-RAND	127	72	2.2×	21.4	0.116	82.5	23.5	0.190	84.1	19.6	0.052	81.2
5	11-1-FAM	127	72	2.2×	22.2	0.134	82.9	24.5	0.211	84.3	20.3	0.067	81.6
6	11-1-EMB	127	72	2.2×	22.0	0.127	82.8	24.2	0.202	84.4	19.8	0.062	81.4
7	11-1-ST	127	72	2.2×	21.8	0.128	82.5	24.0	0.205	84.2	19.9	0.061	81.0
8	10-2	73	73	1.8×	22.3	0.153	83.0	24.3	0.244	84.4	20.5	0.074	81.7
9	10-2-EACH	905	73	1.8×	22.9	0.151	82.6	22.3	0.084	83.4	23.5	0.208	81.9
10	10-2-RAND	183	73	1.8×	23.5	0.162	83.0	25.8	0.240	84.3	21.6	0.094	81.8
11	10-2-FAM	183	73	1.8×	**24.8**	**0.234**	**83.4**	27.0	0.277	84.8	22.9	0.197	82.3
12	10-2-EMB	183	73	1.8×	24.1	0.186	83.2	26.4	0.270	84.6	22.1	0.112	82.1
13	10-2-ST	183	73	1.8×	24.1	0.190	83.1	26.6	0.273	84.3	22.0	0.118	82.0

Table 7.2: Performance comparison over various models on OPUS-100. High and Low denote high-resource language (>= 1 million training sentence pairs) and low-resource language (< 1 million training sample pairs). BS shows the BERTScore [201]. TP and DP indicate the number of parameters loaded in the memory at training and decoding time. For all COMET, BLEU and BERTScore, the higher the better.

Results

From Figure 7.1, we find that for O2M translation, models with 1- or 2-layer decoders have a clear performance drop compared to the standard transformer (6-6). Therefore, our main experiments adopt multiple shallow decoders with 1 and 2 decoder layers. Results on OPUS-100 and TED8 corpora are shown in Table 7.2 and 7.3 respectively.

One language per decoder (EACH) With this assignment method, models obtain superior performance on high resource languages but poor results on low resource languages. On OPUS-100, if each language has its own decoder, we find that it achieves great results on high resource

#	Model	Related						Diverse					
		#TP	#DP	DS	BLEU	COMET	BS	#TP	#DP	DS	BLEU	COMET	BS
1	6-6	64	64	1.0×	**16.7**	**-0.063**	**79.9**	66	66	1.0×	**18.0**	**-0.054**	**81.9**
2	11-1	58	58	2.3×	14.6	-0.330	78.1	61	61	2.6×	15.6	-0.322	80.2
3	11-1-EACH	80	58	2.3×	14.8	-0.312	78.2	83	61	2.6×	16.3	-0.294	80.5
4	11-1-RAND	58	58	2.3×	14.7	-0.304	78.1	74	61	2.6×	15.9	-0.307	80.1
5	11-1-FAM	68	58	2.3×	15.2	-0.262	78.6	74	61	2.6×	16.5	-0.256	80.6
6	11-1-EMB	68	58	2.3×	15.2	-0.262	78.6	74	61	2.6×	16.2	-0.263	80.5
7	11-1-ST	65	58	2.3×	15.0	-0.276	78.3	74	61	2.6×	16.3	-0.272	80.5
8	10-2	59	59	1.9×	15.6	-0.229	78.9	62	62	1.9×	16.9	-0.207	81.0
9	10-2-EACH	104	59	1.9×	16.1	-0.174	79.2	106	62	1.9×	17.2	-0.157	81.1
10	10-2-RAND	78	59	1.9×	16.0	-0.186	79.1	87	62	1.9×	17.3	-0.144	81.6
11	10-2-FAM	78	59	1.9×	16.5	-0.109	79.7	87	62	1.9×	17.7	-0.120	81.5
12	10-2-EMB	78	59	1.9×	16.5	-0.109	79.7	87	62	1.9×	17.2	-0.138	81.5
13	10-2-ST	78	59	1.9×	16.4	-0.123	79.6	87	62	1.9×	17.4	-0.132	81.4

Table 7.3: Translation speed and accuracy trade-off on TED8-Related and TED8-Diverse corpora. BS shows the BERTScore [201]. TP and DP indicate the number of parameters at training and decoding time. For all COMET, BLEU and BERTScore, the higher the better.

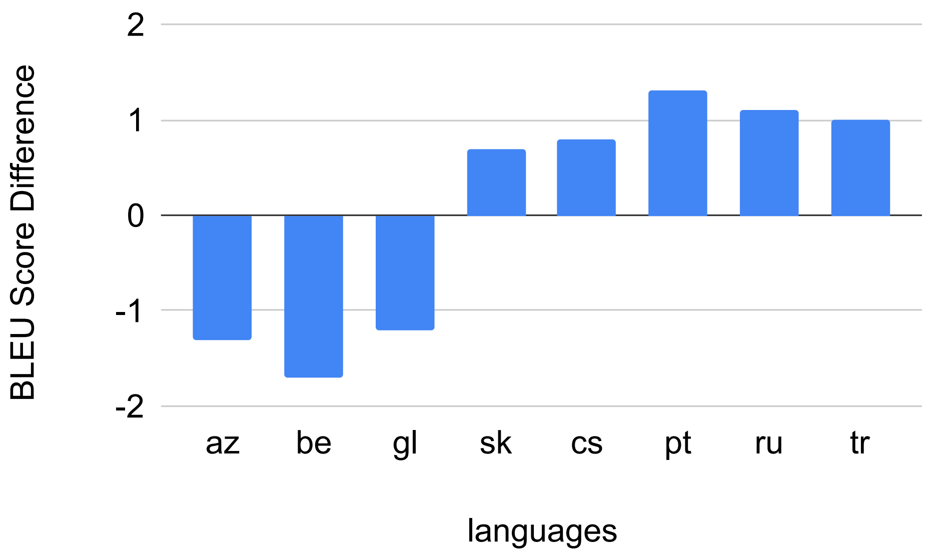


Figure 7.2: The BLEU score difference between models 10-2-EACH and 10-2 on TED8-Related ($\text{BLEU}_{\text{10-2-EACH}} - \text{BLEU}_{\text{10-2}}$). (Left four languages are low-resourced and the right four are high-resourced.)

languages (rows 2 vs. 3 and 8 vs. 9 in Table 7.2). We think that given enough training data, the shallow decoder has enough ability to model one language. However, it performs worse on the low resource languages compared with the baseline (rows 2 vs. 3 and 8 vs. 9 in Table 7.2). To further understand this assignment method, we also show the BLEU score improvement of model 10-2-EACH over 10-2 on TED8-Related in Figure 7.2. The left three languages are relatively low resourced and their performance is lower than the baseline model in which all languages share one decoder[1]. This also demonstrates that their decoders are not able to learn robust representations given a limited amount of training data. And decoders trained with high resource languages generate higher quality translations and we attribute this to enough training data and no negative transfer effect when trained without other languages [4].

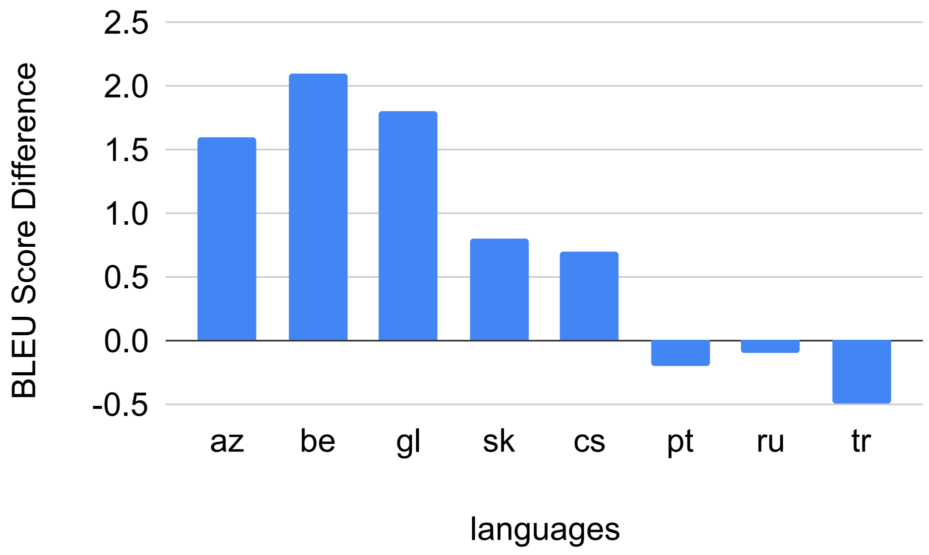


Figure 7.3: The BLEU score difference between models 10-2-FAM and 10-2-EACH on TED8-Related ($\text{BLEU}_{\text{10-2-FAM}} - \text{BLEU}_{\text{10-2-EACH}}$). (Left four languages are low-resourced and the right four are high-resourced.)

Random language set assignment (RAND) We find that the random language set assignment method slightly improves the performance over the baseline due to the sub-optimal knowledge transfer among languages in the same decoder. If each decoder handles a similar number of languages, it also slightly improves the performance compared to the model with one shared decoder (rows 2 vs. 4 and 8 vs. 10 in Tables 7.2 and 7.3). We attribute this to that the shallow decoder performs better given fewer languages. This also demonstrates that one shallow decoder does not have enough capacity to model a large number of languages. However, compared to

[1]Note that although sk is defined as a low resourced language in this dataset, the reason why language sk still have slightly better result is that sk has 61.5k training data but the other three low resource languages (az, be, gl) have less than 10k training sentence pairs.

language family and embedding assignment methods, the random language set method has lower translation quality, showing that how to assign target languages into these decoders is also crucial.

One language family per decoder (FAM) We group all languages into several groups according to their language families and assign each family to one shallow decoder. As a result, we have 14, 4 and 5 language families in OPUS-100, TED8-Related and TED8-Diverse corpora respectively. From the comparison between rows 2 vs. 5 and 8 vs 11 in Tables 7.2 and 7.3. It is clear to find that language family-based decoders achieve better accuracy and maintains low latency at the same time. Furthermore, models with multiple 2-layer decoders on OPUS-100 achieve better performance than the model 6-6 and obtain around a 1.8 times speedup at inference time. We think the improvement is mainly coming from the better knowledge transfer among similar languages (in one language family) and enough capacity for each decoder to handle a subset of languages.

In order to understand this further, we plot the BLEU score difference between models 10-2-EACH and 10-2-FAM on TED8-Related in Figure 7.3. We find that the major improvement of model 10-2-FAM over 10-2-EACH is from the low resource languages which means the high resource languages help their relevant low resource languages effectively.

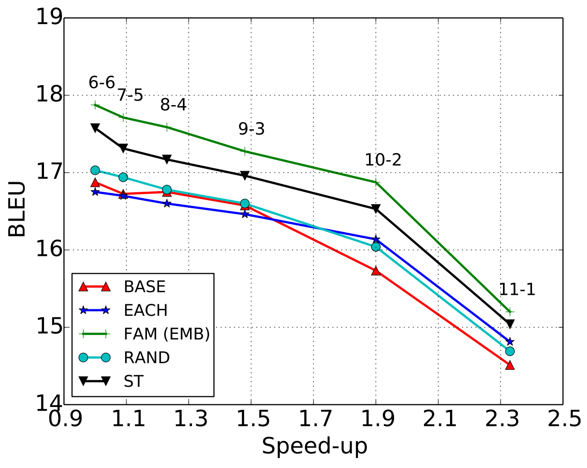
Language embedding-based assignment (EMB) For a fair comparison, languages are also grouped into the same number of language families according to language embeddings from the well-trained baseline model 6-6. Grouping results are listed in the Appendix 7.7. We first find that the language embedding-based grouping method is able to group similar languages together, showing the ability of language embeddings to effectively capture language characteristics during training. For example, on TED8-Related, the language embedding achieve the same grouping result as the language family-based one shown in Table 7.1. The language embedding-based assignment method achieves similar results compared to the language family-based one and effectively improves the performance of the baseline model.

Self-taught language assignment (ST) In this method, the model tries to assign target languages to multiple decoders automatically and there is no need to have any prior knowledge (linguistic families) or well-trained models (language embeddings). From the rows 7 vs. 2 and 13 vs. 8 in Tables 7.2 and 7.3, our self-taught method significantly better than the corresponding baseline. It also achieves similar results compared with the language family (embedding)-based language assignment methods, demonstrating the effectiveness of this method.

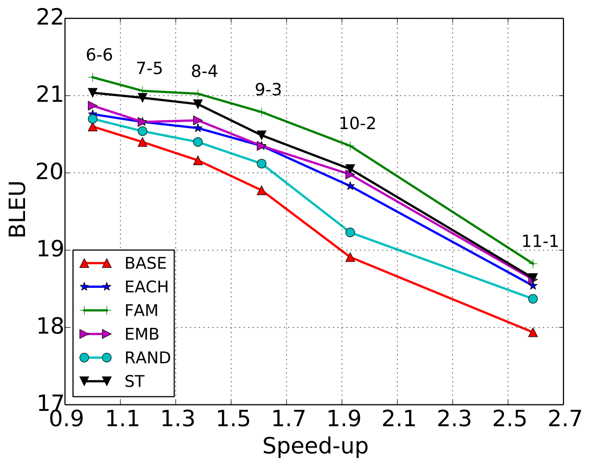
Statistic Significant Test Finally, we try to compare our DEMSD-FAM model with the standard transformer model (6-6). We employ `comet-compare` to get statistical significance with

Paired T-Test and bootstrap resampling [88]. On OPUS-100, our 10-2 DEMSD model with language family assignment method manages to surpass the transformer model. However, on TED8 corpora, our DEMSD still underperforms the 6-6 model. We hypothesize that with more languages, the capacity issue of the shallow decoder is more severe so that the improvement of our DEMSD model on OPUS-100 is more outstanding. In this next chapter, we will seek to close this performance gap.

7.5 Analysis and Discussion



(a) TED8-Related



(b) TED8-Diverse

Figure 7.4: Multiple decoders with various layer allocations of Transformer on TED8 datasets. X-Y denotes X and Y layers in the encoder and decoder respectively. 'BASE' denotes the shared decoder model.

Multiple decoders for various layer allocations

In our main experiments, we use multiple very shallow decoders (i.e., 1 and 2-layer decoders) because there is a clear performance drop when using a single decoder with this configuration for one-to-many translation compared with the standard transformer (6-6), and compared to deeper decoders, employing multiple 1- or 2-layer decoders keeps the number of parameters manageable at training time. Nevertheless, it will be meaningful to explore the effect of multiple decoders on various layer allocations. Considering the model size and tractable training time, we only conduct experiments on TED8 corpora and the results are shown in Figure 7.4. On each line (the same language assignment method), the deeper decoders achieve better performance and the shallower decoder has lower latency. Moreover, if we compare language family-based assignment and the baseline models, given the same decoding speed at inference time, the former consistently improves the performance with the same decoding speed at inference time. And with similar performance, e.g., 10-2-FAM and 6-6, our best multiple shallow decoder models have much lower latency.

Speed-accuracy trade-off in multilingual machine translation

From the above experiments and findings, in the one-to-many translation, the DESD framework obtains a superior speed-accuracy trade-off. For example, the model with 10 encoder layers and 2 decoder layers obtains better translation quality and a 1.8$\times$ speedup.

Under the one-to-many setting, multiple shallow decoders are needed to mitigate the performance drop of the DESD model. And the crucial part is to group languages with similar features to one decoder to obtain better knowledge transfer among languages (our FAM, EMB and ST methods). With this, our DEMSD model with multiple 2-layer decoders is capable of achieving similar performance and a 1.8$\times$ speedup compared to the standard transformer.

7.6 Summary

In this chapter, we study speed-accuracy trade-offs using various layer configurations for multilingual neural machine translation. We find that for many-to-one translation, deep encoder and shallow decoder (DESD) models improve decoding speed while maintaining translation quality with the same model capacity. However, for one-to-many translation we observe a drop in quality when the decoder depth is reduced. To mitigate the performance drop of DESD models in one-to-many translation, we propose using a shared encoder and *multiple* shallow decoders (DEMSD). Our best DEMSD models with 2-layer decoders are capable of speeding up decoding by 1.8 times while achieving the competitive quality compared to a standard transformer.

7.7 Appendix

Training details of DESD model

In order to explore how DESD models work on multilingual machine translation, we train transformer-based models with various layer allocations on three multilingual machine translation corpora, OPUS-100, TED8-Related and TED8-Diverse. For a fair comparison, the training process is the same across all models.

Language Family	Languages
INDO-IRANIAN	hi, hr
SLAVIC	mk, bs, bg
KOREAN	ko
HELLENIC	el
ROMANCE	fr

Table 7.4: Language families in the TED8-Diverse corpus.

On OPUS-100, we employ the standard transformer-base model: 8 attention heads per layer, 512 model dimensions, 2048 hidden dimensions and 0.1 dropout. All models are trained for 100,000 with batches of 64k tokens using Adam and 0.1 label smoothing. The learning rate goes to $4e{-4}$ within 4,000 steps,and then decays with the inverse square-root schedule.

On TED8 corpora, following [184], a smaller transformer model is adopted, i.e., 4 attention heads per layer, 512 model dimensions, 1024 hidden dimensions and 0.3 dropout. All models are trained for 40,000 with batches of 16k tokens using Adam and 0.1 label smoothing. The learning rate goes to $2e{-4}$ within 4,000 steps, and then decays with the inverse square-root schedule.

Language family assignment results

In Table 7.4, we show the language family-based assignment result on TED8-Diverse. Since this corpus is collected without considering relatedness, some groups just have one language. But its multiple decoders model improves the accuracy, showing the effectiveness of this method.

The language families in OPUS-100 is shown in Table 7.5.

Language embedding assignment results

On TED8-Related, we obtain the same language assignment results as the language family-based one. On TED8-Diverse, the result of the language embedding assignment method is pretty similar to the language family assignment result (Table 7.6. The only difference is that language bg is grouped with language mk. We think this is because the language embedding not only contains the linguistic feature but the data feature as well.

Language Family	Languages
AFRO-ASIATIC	am, ar, ha, he, mt
AUSTROASIATIC	km, vi, ld, mg, ms
CELTIC	br, cy, ga, gd
DRAVIDIAN	kn, ml, ta, te
GERMANIC	af, de, fy, is, li, nb, nl, nn, no, sv, yi
INDO-ARYAN	as, bn, da, gu, hi, mr, ne, pa, si, ur
IRANIAN	fa, ku, or, ps, tg
NIGER–CONGO	ig, rw, xh, zu
OTHER	el, eo, eu, ja, ka, ko, my, sq, th, zh
ROMANCE	ca, es, fr, gl, it, oc, pt, ro, wa
SLAVIC	mk, pl, ru, sh, sk, sl, sr, uk
TURKIC	az, kk, ky, tk, tr, tt, ug, uz
URALIC	et, fi, hu, se

Table 7.5: Language families in the OPUS-100 corpus.

Group Id	Languages
0	hi, ko
1	mk, bg
2	ko, bs
3	el
4	fr

Table 7.6: Language embedding-based language assignment result on the TED8-Diverse corpus.

Chapter 8

Deep encoder, Shallow Decoder with Language-level Mixture-of-Experts

In the previous chapter, we propose a deep encoder, multiple shallow decoders (DEMSD) architecture to mitigate the capacity bottleneck and successfully boost the performance of the vanilla deep encoder, shallow decoder (DESD) model while maintaining its high decoding speed on the one-to-many translation task. In this chapter, we investigate mixture-of-experts (MoEs), a successful technique to scale transformer models, to increase the shallow decoder's capacity. We further explore routing strategies from various granularity to enjoy the benefit of sparsely activated sub-networks at inference time for memory efficiency. Experimental results show that our best model surpasses the DEMSD model under the same capacity budget. An empirical study is conducted to show that increasing the capacity of the feed-forward network layer is more effective to improve the translation quality than other components in a shallow decoder block. Furthermore, our best model achieves comparable or better translation accuracy compared to the standard transformer model with a $1.8\times$ speed-up at inference time.

8.1 Introduction

The deep encoder, shallow decoder (DESD) architecture successfully accelerates the decoding process without sacrificing the superior performance compared to the standard transformer model on bilingual and many-to-one translation tasks [73, 92]. However, in our previous chapter, we find that although shallow decoder-based models have higher decoding speed, there is an apparent performance drop compared to the standard transformer model since the shallow decoder does not have enough capacity to model several different target languages adequately. With the goal of obtaining low latency while maintaining translation quality, in Chapter 7, we propose using a deep encoder, multiple shallow decoders (DEMSD) architecture where each decoder is responsible for a subset of target languages. Several language assignment methods are explored to assign each target language (or language group) to one of these multiple decoders. Our best DEMSD model successfully boosts the performance of the vanilla DESD architecture. However, our DEMSD model still underperforms the standard transformer model (on TED8-Related and Diverse). Therefore, in this chapter, we aim to close this performance gap while maintaining the latency advantage. Recently, sparse mixture-of-experts (MoEs) [40, 104, 157] has been a successful technique for scaling transformers without a proportional increase in training computation. In this chapter, we study the effect of MoEs on the shallow decoder.

Different from previous works [40, 104, 157] which aim to increase the capacity of the whole model (both encoder and decoder), we mainly focus on the decoder side because we already have a deep encoder. Additionally, in the conventional MoEs, each token in the input sequence chooses a (distinct) subset of experts independently so that the computational cost is just proportional to the size of the selected experts. However, this routing strategy may harm inference efficiency in terms of memory consumption. More specifically, since each token will be routed to a subset of experts independently, a long sequence consisting of lots of tokens will expand a wide range of experts. In practice, we have to load all experts into the memory during inference which may surpass the memory limitations and become prohibitive for inference. Therefore, in this chapter, we explore alternative routing methods which are trained to leverage information from different granularity (language and language family) to direct all tokens in a language to the same subset of experts. Consequently, during inference time, given a specific target language, only a fixed subset of experts corresponding to this language is needed to load. Experimental results on three multilingual translation benchmarks show the effectiveness of our methods to improve translation accuracy over the DESD model and successfully close the performance gap with the standard transformer model while maintaining the high decoding speed.

Furthermore, when comparing to DEMSD models, with the same capacity budget, MoEs-based models perform better and we find that the major difference between them is that MoEs-based models allocate more capacity to feed-forward network layers instead of all components simultaneously. To understand this finding, we revisit the capacity bottleneck issue of the shallow decoder on the one-to-many translation task through an empirical analysis. Specifically, we focus on three main components in a transformer decoder block (Figure 8.3), i.e., self-attention, cross-attention (encoder-decoder attention) and feed-forward network. Through improving the capacity of all possible combinations of these three components and comparing the translation quality, we find that the feed-forward network layer is the key to mitigating the capacity bottleneck and merely increasing its capacity can largely boost the performance.

8.2 Background and Related Work

Mixture of Experts (MoEs)

MoEs is first proposed in Jacobs et al. [66] as an ensemble method of multiple individual models and formulated as:

$$y = \sum_{i=1}^{N} g(x)_i f_i(x) \tag{8.1}$$

where $\sum_{i=1}^{N} g(x)_i = 1$. $g(x)_i$ indicates the probability of expert $f_i(x)$ and N is the number of expert networks. $g(\cdot)$ is also called the gating network which produces a distribution over the N experts based on the input and the final output y is the weighted sum of outputs from all experts.

Shazeer et al. [157] use MoEs as a basic component of the neural network so that it can accept the output of the previous layer as input and output to the following layers, in an end-to-end training style. When applying MoEs to transformer [40, 104], each expert is a two-layer neural network with a ReLU activation function. Moreover, for each input example, the model is able to select only k experts with top-k probability scores to save computations by the gating network conditioned on the input. This can be regarded as a form of conditional computation [12]. Therefore, the advantage of sparse MoEs is to increase model capacity without a proportional increase in computational costs.

The gating network is vital in the routing process which is modeled by a softmax activation function to indicate the weights of each expert in processing incoming tokens. However, it is common to have a loading imbalance issue [104]: the majority of tokens will be guided to just a small subset of experts. This would result in a capacity overflow for only a few experts and under-utilization for the remaining ones, leading to a suboptimal solution. In Lepikhin et al. [104], Shazeer et al. [157], they design an auxiliary loss to give a penalty on this issue. More details and discussions could be found in Lepikhin et al. [104], Shazeer et al. [157].

Besides works mentioned above, recent work has implemented sparse routing via k-means clustering [53], linear assignment to maximize token-expert affinities [107], tasking specific routing [96] or hashing [107].

In this chapter, we mainly employ MoEs to add the shallow decoder's capacity and make use of multilingual characteristics to explore various routing strategies.

8.3 Improving the Capacity of the Shallow Decoder via Mixture-of-Experts

Sparsely-gated Mixture-of-Experts (MoEs)

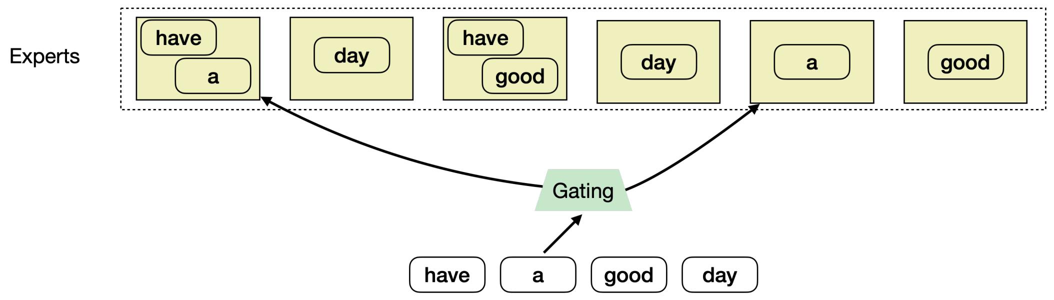
Although MoEs has been applied to multilingual machine translation [104, 157], its goal is to scale up the capacity of the feed-forward network layer in both the encoder and decoder and a large number of experts, such as 2048, are used which will incur a huge number of parameters. Different from them, in this chapter, we solely employ MoEs to improve the capacity of the FFN layer in the shallow decoder. More importantly, as discussed above, the conventional routing approach treats tokens in the input sequence independently, which may fail the memory requirement during inference. Therefore, in the chapter, we investigate several routing strategies from different granularity (token, language, language-family), which are described below.

Token-level MoEs (T-MoEs): It is the default routing strategy for MoEs in which input tokens will go through the gating network independently and choose (various) subsets of experts. From the practical standpoint, the model becomes prohibitive large at decoding time because we need to load all experts into the memory which may be beyond the memory requirement. To enjoy the benefit of sparsely activated sub-networks at inference time, i.e., extracting out a sub-network to decode for a particular target language, two routing strategies are discussed below.

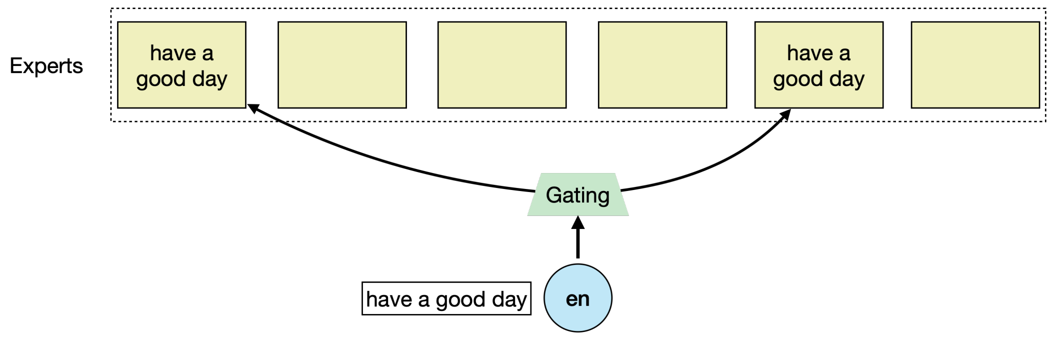
Language-level MoEs (L-MoEs): Given an input sequence belonging to a target language l, the input to the gating network $g(\cdot)$ is the embedding of the language l and the gating network dispatches this language to a subset of experts, hence, all tokens in the same language will be routed to the same subset of experts. As a result, during decoding time, given a specific target language, the model only needs to pre-load experts which are selected by this language to avoid large memory consumption. However, we find that this strategy suffers from the loading imbalance issue which makes the full capacity underutilized. To mitigate it, we introduce a constraint on the routing function which is described below.

Language-level MoEs with Linguistic Family Constraint (LF-MoEs): In this routing strategy, we first group target languages into several clusters according to their linguistic families, then all experts are split into several subsets evenly, the size of which equals the number of language family groups. Each language can only select experts which are assigned to its language family.

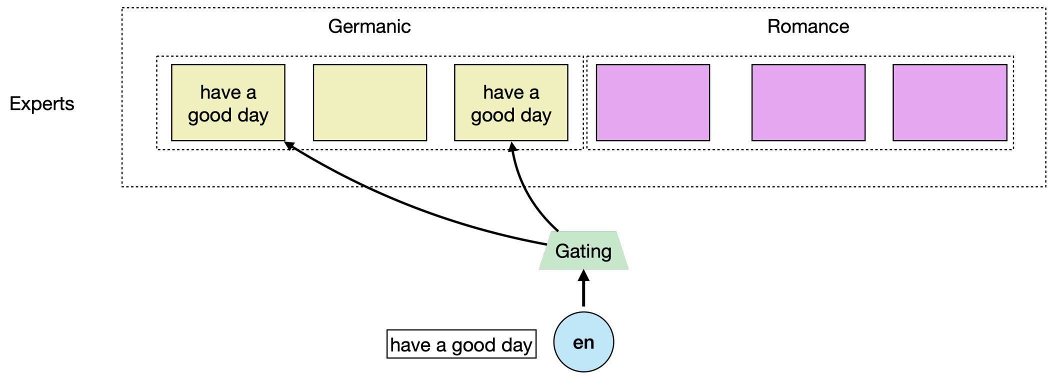
This constraint can be regarded as a predefined loading balance rule. With this constraint, the number of experts which can be selected by a given language is smaller than other routing strategies. If top-k weight experts will be chosen for a given input, the extreme case is that there are only k experts in each group and we consider it as a deterministic MoEs routing strategy since there is no need to select a sparse combination of experts.



(a) Token-level MoEs



(b) Language-level MoEs



(c) Language-level MoEs with the Linguistic Family Constraint

Figure 8.1: MoEs with various routing strategies. Each expert is a feed-forward network block in transformer.

We illustrate their differences in Figure 8.1. Figure 8.1a shows the token-level MoEs. Each token will choose top-k (k=2 in the case) experts independently. As a result, the gating result for the input sequence may span a wide range of experts. On the contrary, language-level MoEs (Figure 8.1b) route all tokens in the input sequence to the same subset of experts according to the language information. Figure 8.1c refers to the language-level MoEs with the linguistic family constraint. For a target language, it can only choose experts belonging to its linguistic family. In this example, the target language is English which is a Germanic language. As a result, it can only choose top-k Germanic experts.

8.4 Experiments

Experiment Settings

Same as the previous chapter, we also conduct experiments on three multilingual translation benchmarks mentioned above, i.e., TED8-Related, TED8-Diverse and OPUS-100. In MoEs models, the feed-forward network (FFN) layer in the shallow decoder is replaced with multiple parallel FFN experts. For a fair comparison to DEMSD, in our main experiments, 12, 15 and 28 experts are employed for MoEs models on TED8-Related, TED8-Diverse and OPUS-100 corpora respectively to make the model capacity (number of parameters) the same as DEMSD models. Following previous works [104, 157], the gating network chooses top-2 weight experts for each token or language so only 2 experts are needed during inference time for a given target language. For our language-level MoEs with the linguistic family constraint model, there are 3 and 2 experts in each language family group on TED8-Corpora and OPUS-100 datasets respectively. Here we show the language family group of the TED8-Related dataset in Table 8.1 and the language family information of other datasets can be found in Chapter 7[1].

Language Family	Languages
TURKIC	az, tr
SLAVIC	be, ru
ROMANCE	gl, pt
CZECH–SLOVAK	sk, cs

Table 8.1: Language families in the TED8-Related corpus.

On OPUS-100, we follow most of the standard hyperparameters in the transformer-base [178]: 8 attention heads per layer, 512 model dimensions, 2048 hidden dimensions and 0.1 dropout. We train batches of 128k tokens using Adam [81] with β = (0.9, 0.98) and ϵ = 10^{-6} and 0.1 label smoothing. Following [199], the learning rate goes to $4\text{e}{-4}$ within 4,000 steps, and then decays with the inverse square-root schedule. All models are trained for 300,000 steps. Furthermore, to mitigate the training data imbalance issue, the temperature sampling method is adopted [4]

[1]There are 4, 5, 14 language families in TED8-Related, TED8-Diverse and OPUS-100 corpora respectively.

which is set as 5 in all experiments. On TED8 corpora, a smaller transformer model with 512 model dimensions, 1024 hidden dimensions and 0.3 dropout is adopted. All models are trained for 40k steps with batches of 16k tokens with a smaller learning rate 2e−4. The other training procedure is the same as the OPUS-100.

For deep encoder, shallow decoder models, we apply the proposed methods to shallow decoders with 1 and 2-layers. During decoding, we use beam search with beam size 4 and length penalty 0.6. We employ BLEU [133], BERTScore [201] and COMET (wmt20-comet-da) [147] to assess the translation quality. Furthermore, we employ `comet-compare` to compare various models to get statistical significance [88] when necessary. For the evaluation speed, **DS**, it is measured by the number of tokens the system translates per second given one sentence at a time on a single GPU.

For token- and language-level MoEs, an auxiliary loss mentioned in Shazeer et al. [157] is employed to mitigate the loading imbalance issue. For language-level MoEs with the linguistic family constraint, we regard this constraint as a predefined loading rule so we do not apply any auxiliary loss.

ID	Model	TED8-Related						TED8-Diverse					
		TP	DP	Speed	BLEU	COMET	BS	TP	DP	Speed	BLEU	COMET	BS
1	6-6	64	64	1.0×	16.7	-0.063	79.9	68	68	1.0×	18.0	-0.054	81.9
2	DEMSD (11-1)	68	58	2.3×	15.2	-0.262	78.6	74	61	2.6×	16.5	-0.256	80.6
3	DESD (11-1)	58	58	2.3×	14.6	-0.330	78.1	61	61	2.6×	15.6	-0.322	80.2
4	+ T-MoEs	66	66	2.1×	15.6	-0.248	78.8	74	74	2.3×	16.7	-0.198	80.9
5	+ L-MoEs	68	59	2.3×	15.2	-0.226	78.9	74	62	2.6×	16.3	-0.259	80.7
6	+ LF-MoEs	68	59	2.3×	15.9	-0.239	78.1	74	62	2.6×	17.3	-0.177	81.2
7	DEMSD (10-2)	78	59	1.9×	16.5	-0.109	79.7	87	62	1.9×	17.7	-0.120	81.5
8	DESD (10-2)	59	59	1.9×	15.6	-0.229	78.9	62	62	1.9×	16.9	-0.207	81.0
9	+ T-MoEs	78	78	1.8×	16.8	-0.087	79.5	87	87	1.8×	17.6	-0.122	81.4
10	+ L-MoEs	78	61	1.9×	16.2	-0.138	79.4	87	64	1.9×	17.4	-0.128	81.2
11	+ LF-MoEs	78	61	1.9×	17.3	-0.064	79.7	87	64	1.9×	18.1	-0.060	81.9

Table 8.2: Performance comparison over various models on TED8-Related and TED8-Diverse datasets. BS (BERTScore), TP (# Training Parameters), DP (# Decoding Parameters) show the BERTScore scores and number of parameters loaded in the memory during training and inference time respectively. For all COMET, BLEU and BERTScore, the higher the better.

Experiment Results

Our main results are shown in Table 8.2 and 8.3.

ID	Model	TP	DP	Speed	All BLEU	All COMET	All BS	Low BLEU	Low COMET	Low BS	High BLEU	High COMET	High BS
1	Trans. Base	77	77	1.0×	23.2	0.186	83.1	25.7	0.265	84.6	21.1	0.117	81.8
2	DEMSD (11-1)	127	72	2.2×	22.6	0.134	82.9	24.9	0.211	84.3	20.7	0.067	81.6
3	DESD (11-1)	72	72	2.2×	21.1	0.060	82.5	23.2	0.168	84.0	19.3	-0.033	81.1
4	+ T-MoEs	127	127	2.0×	23.0	0.128	82.8	25.4	0.212	84.3	20.8	0.055	81.6
5	+ L-MoEs	127	74	2.2×	22.4	0.114	82.8	24.6	0.195	84.2	20.6	0.044	81.5
6	+ LF-MoEs	127	74	2.2×	23.3	0.146	83.0	25.7	0.229	84.4	21.2	0.075	81.7
7	DEMSD (10-2)	183	73	1.8×	24.8	0.234	83.4	27.0	0.277	84.8	22.9	0.197	82.3
8	DESD (10-2)	73	73	1.8×	22.3	0.153	83.0	24.3	0.244	84.4	20.5	0.074	81.7
9	+ T-MoEs	183	183	1.7×	24.9	0.240	83.4	27.3	0.296	84.8	22.9	0.192	82.2
10	+ L-MoEs	183	77	1.8×	24.5	0.228	83.4	26.8	0.281	84.8	22.5	0.182	82.3
11	+ LF-MoEs	183	77	1.8×	25.5	0.252	83.6	27.8	0.302	85.0	23.5	0.209	82.4

Table 8.3: Performance comparison over various models on OPUS-100. High and Low denote high-resource language (>= 1 million training sentence pairs) and low-resource language (< 1 million training sample pairs). For all COMET, BLEU and BERTScore, the higher the better.

In terms of translation quality Compared to the vanilla deep encoder, shallow decoder (DESD) model, our MoEs-based models from all granularity (token, language, language family) achieve consistent gains across all datasets. Among various routing strategies, we see that the language-level MoEs with the linguistic family constraint (LF-MoEs) achieve the best performance. We attribute it to the maximum positive knowledge transfer by grouping languages belonging to the same language family together with the predefined loading balancing rule. This observation is consistent with findings in Chapter 7. Our Language-level MoEs (L-MoEs) suffers from the loading imbalance issue even with an auxiliary loading balance loss. For instance, on TED8-Related dataset, 4 out of 12 experts are not selected by any language. Therefore, the capacity is under-utilized and improvement over the baseline is smaller. Compared to the DEMSD model, our LF-MoEs model is significantly better through statistical significance testings (`comet-compare`). Furthermore, in contrast to the transformer base model (row 1), no significant difference has been found for our best model (10-2 + LF MoEs) on TED8 corpora and our (10-2 + LF MoEs) model is significantly better on OPUS-100 benchmark.

In terms of translation speed Shallow decoder-based models have a clear advantage over the transformer-base model. Specifically, 1- and 2-layers decodes are able to achieve 1.8× and 2.2× speedups respectively. For token-level MoEs, it has a slightly lower speed compared to language-level MoEs models since much more experts are involved in the computation process.

In terms of model size efficiency at inference time For language-level MoEs (with the linguistic family constraint), given a specific target language, during the inference process, it only needs to pre-load top-2 weight experts so only a small number of extra parameters are introduced. For example, compared to the DESD model, language-level MoEs models introduce 3%∼5% extra parameters. However, token-level MoEs incurs more than 80% parameters at inference time.

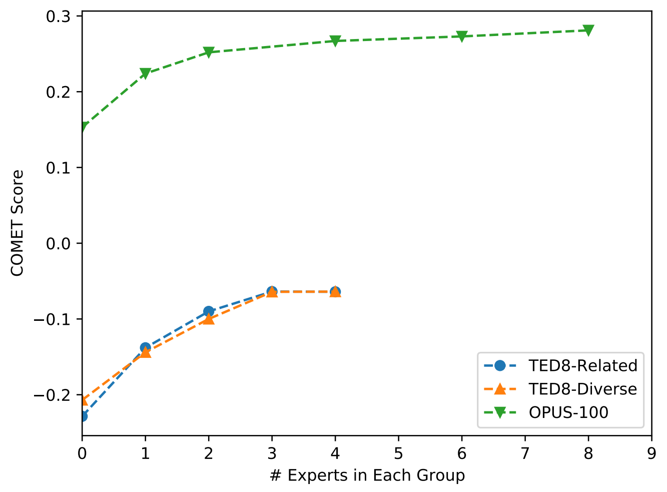


Figure 8.2: Translation quality of models with various number of experts in each language family group. '0' denotes no MoEs and all languages share the same feed-forward network.

Number of Experts in Each Language Family Group

In our main experiments, the number of experts is set for the purpose of a fair comparison to the DEMSD model. Here, we explore models with different numbers of experts on those three benchmarks for our 10-2 LF-MoEs model. The translation quality in COMET with respect to the number of experts in each language family group is shown in Figure 8.2. We find that with more experts, the better translation quality will be obtained but the improvement becomes smaller.

8.5 Revisit Capacity Bottleneck of the Shallow Decoder

In our main experiments, we find that our LF-MoEs-based models outperform DEMSD models under the same capacity budget. The major difference between these two models is that MoEs models mainly allocate the capacity to the FFN layer but DEMSD tries to add capacity to all components in the decoder simultaneously. To understand this difference, in this section, we revisit the capacity bottleneck issue. Specifically, we first review the basic structure of the transformer decoder block and identify three key modules in it. Then we conduct an empirical study to analyze the capacity bottleneck of the shallow decoder.

Transformer Decoder Block

The transformer decoder consists of a stack of identical blocks and one block is shown in Figure 8.3. We can see that there are typically three stages in a standard transformer decoder block: self-attention, cross attention (encoder-decoder attention) and feed-forward network (FFN). All of these modules are followed by layer normalization [5] and a residual connection [62]. Given an input sequence, the standard self attention allows each position in the decoder to attend to all positions in the decoder up to and including that position. To preserve the autoregressive property of the language modelling, the attention matrix is causally masked to prevent future information leak to the past. The next component is cross attention, a.k.a. encoder-decoder attention. This allows every position in the decoder to attend over all positions in the source sequence to obtain the alignment between the target and source sequences. The last stage is a position-wise feed-forward network (FFN), which consists of two linear transformations with a default ReLU activation function in between. This FFN module applies to each position separately and identically.

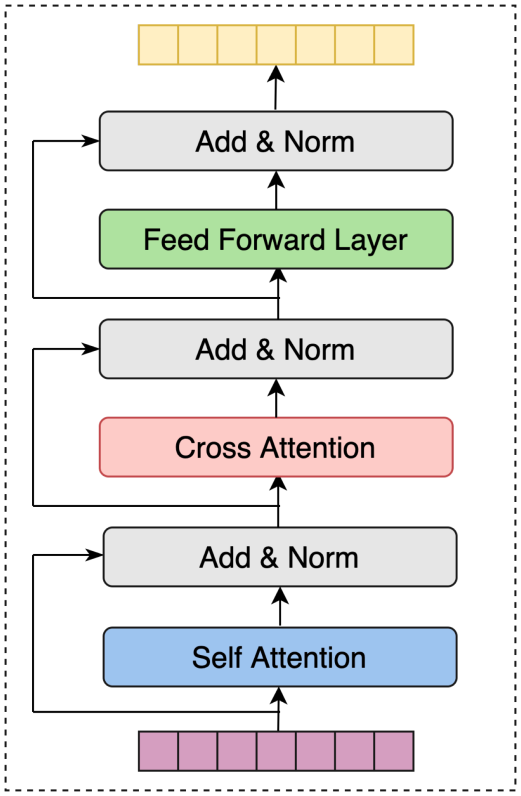
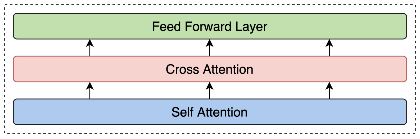


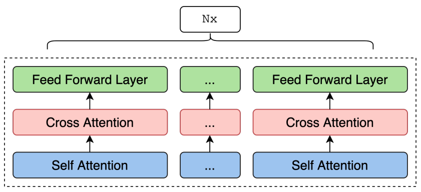
Figure 8.3: Transformer decoder block.

Capacity Bottleneck Breakdown

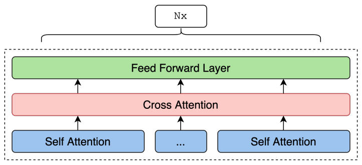
In our proposed multiple shallow decoders-based models, DEMSD (Chapter 7), we have multiple parallel decoders and each target language will be routed to a specific decoder according to various assignment methods. In another word, DEMSD can also be regarded as a giant decoder, in which there are multiple independent sub-networks (decoders) (Figure 8.4b) and the capacity of all components in the decoder is improved at the same time.



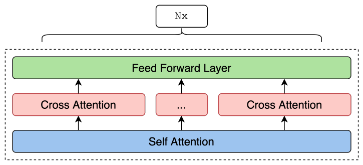
(a) No Capacity Increase (standard)



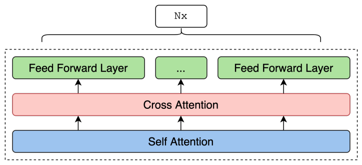
(b) All Capacity Increase (DEMSD)



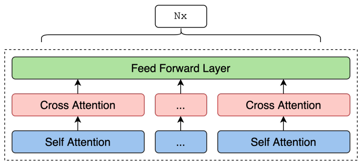
(c) SA Capacity Increase



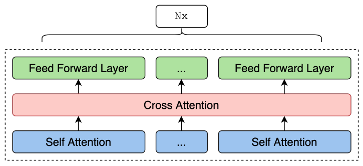
(d) CA Capacity Increase



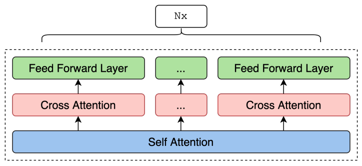
(e) FFN Capacity Increase



(f) SA+CA Capacity Increase



(g) SA+FFN Capacity Increase



(h) CA+FFN Capacity Increase

Figure 8.4: Increasing the capacity of various combinations of components in a transformer decoder block. Layer normalization and the residual connection are ignored for simplicity. SA, CA and FFN denote the self-attention, cross attention and feed-forward networks respectively.

Here we conduct an ablation study to understand the capacity bottleneck of the shallow decoder and which stage in the decoder needs to have a large capacity. Specifically, we focus on the breakdown of the decoder block and explore the capacity bottleneck implications at the component level via increasing the capacity of different subsets of these components. For example, in Figure 8.4d, we only increase the capacity of the cross attention parts through employing multiple parallel cross attention modules. Each target language will be assigned to one of them. In this case, all target languages will share the same self attention and feed-forward network. All configurations are shown in Figure 8.4. Experiments are conducted on three datasets mentioned above which are OPUS-100, TED8-Related and TED8-Diverse.

Following Chapter 7, 11-1 (11-layer encoder and (multiple) 1-layer decoder(s)) and 10-2 architectures are explored. We employ the linguistic family-based assignment methods. The same model configuration and training details mentioned above are employed. Average BLEU [133], COMET (wmt20-comet-da) [147] and BERTScore [201] over all language directions are computed to assess the translation quality.

ID	Model	11-1-FAM			10-2-FAM		
		BLEU	COMET	BS	BLEU	COMET	BS
1	No	14.6	-0.329	78.1	15.6	-0.229	78.9
2	All	15.2	-0.262	78.6	16.5	-0.109	79.7
3	SA	14.5	-0.340	78.0	15.6	-0.220	78.9
4	CA	14.8	-0.280	78.5	16.1	-0.137	79.5
5	FFN	15.1	-0.263	78.6	16.2	-0.119	79.5
6	SA + CA	14.7	-0.291	78.3	15.9	-0.148	79.4
7	SA + FFN	14.9	-0.276	78.4	16.4	-0.110	79.6
8	CA + FFN	15.2	-0.261	78.6	16.4	-0.107	79.5

Table 8.4: Comparison between models with various capacity increase methods on TED8-Related. BS denotes BERTScore. For all COMET, BLEU and BERTScore, the higher the better.

ID	Model	11-1-FAM			10-2-FAM		
		BLEU	COMET	BS	BLEU	COMET	BS
1	No	15.6	-0.322	80.2	16.9	-0.207	81.0
2	All	16.5	-0.256	80.2	17.7	-0.120	81.5
3	SA	15.3	-0.347	80.0	15.6	-0.219	80.4
4	CA	15.9	-0.290	80.4	17.3	-0.147	81.4
5	FFN	16.3	-0.251	80.7	17.3	-0.134	81.3
6	SA + CA	15.7	-0.302	80.2	17.3	-0.161	81.3
7	SA + FFN	16.0	-0.274	80.5	17.5	-0.128	81.6
8	CA + FFN	16.5	-0.248	80.7	17.6	-0.125	81.5

Table 8.5: Comparison between models with various capacity increase methods on TED8-Diverse. BS denotes BERTScore. For all COMET, BLEU and BERTScore, the higher the better.

Results on TED8 corpora are shown in Tables 8.4 and 8.5 (OPUS-100 results are shown in the Appendix). The vanilla shallow decoder without any capacity increase (NO) performs much worse than DEMSD model (ALL) which increases the capacity of all modules (row 1 vs. 2). When we increase the capacity of only one module (rows 3, 4, 5), through statistic significance tests (`comet-compare`), models with a large capacity feed-forward network (row 5) perform significantly better than adding the capacity to SA or CA. Furthermore, when we increase the capacity of two modules (rows 6, 7, 8), there is a clear performance difference between models with and without capacity-enhanced feed-forward networks (row 6 vs. 7, 8). Similar observations are made on the OPUS-100 dataset. Therefore, from this empirical study, we see that having a large capacity feed-forward network is key to mitigating the capacity bottleneck issue. Therefore, MoEs which allocates capacity to FFN layers can achieve great performance.

8.6 Summary

In this chapter, we investigate mixture-of-experts to increase the capacity of the shallow decoder while maintaining its high decoding speed. After exploring routing strategies from various granularity (token, language and language family), we empirically show that language-level MoEs with the linguistic family constraint models outperforms DEMSD on three multilingual neural machine translation benchmarks. Furthermore, we conduct an empirical study to revisit the capacity bottleneck issue of the shallow decoder and find out that adding more capacity to the FFN layer of the shallow decoder is key to boosting the translation quality. Finally, in contrast to the standard transformer model, our best model can successfully close the performance gap while achieving 1.8$\times$ speed up on average at the same time.

8.7 Appendix

Capacity Bottleneck Results on OPUS-100 Corpora

We show capacity bottleneck experiment results on TED8-Related and -Diverse datasets. Compared to OPUS-100 results, similar observations are found. Capacity-enhanced FFN models effectively improve the performance of DESD models.

ID	Model	11-1-FAM			10-2-FAM		
		BLEU	COMET	BS	BLEU	COMET	BS
1	No	21.1	0.060	82.5	22.3	0.153	83.0
2	All	22.6	0.134	82.9	24.8	0.234	83.4
3	SA	21.6	0.069	82.5	23.2	0.166	83.0
4	CA	21.2	0.062	82.4	23.0	0.181	83.1
5	FFN	22.7	0.123	82.8	24.4	0.224	83.4
6	SA + CA	21.7	0.077	82.5	23.3	0.185	83.2
7	SA + FFN	23.0	0.123	82.8	24.7	0.230	83.4
8	CA + FFN	22.4	0.114	82.8	24.6	0.228	83.4

Table 8.6: Comparison between models with various capacity increase methods on OPUS-100. BS denotes BERTScore. For all COMET, BLEU and BERTScore, the higher the better.

Chapter 9

Conclusion and Future Directions

This thesis tries to improve the decoding efficiency of neural machine translation (NMT) from three perspectives, optimizing the complexity of modules in NMT, improving decoding parallelizability and allocating the capacity of multilingual neural machine translation models.

Our contributions are summarized as follows:

- We aim to optimize the vocabulary representation to reduce the parameter size and computation complexity of the embedding and softmax layers. Two word encoding mechanisms are investigated and we find that Byte Pair Encoding achieves the best efficiency-performance trade-off due to its strong ability to handle rare and out-of-vocabulary words. Therefore, we suggest readers build NMT systems with subword-based vocabulary.
- We develop Luna, a simple, efficient and effective linear attention mechanism used as a drop-in substitute for the standard attention mechanism. By introducing an extra input with a fixed length, Luna is capable of capturing adequate contextual information while reducing the computational complexity of attention operations from quadratic to linear. Experimental results indicate the advantage of Luna in translating long sentences in terms of memory consumption and latency. We find that Luna is capable of achieving superior performance on various natural language and vision understanding tasks such as sentiment analysis, natural language inference, image classification, etc. On language generation tasks such as machine translation, the efficiency advantage of Luna become clear on *long* sequences. Therefore, we encourage readers to apply Luna to other NLP and vision tasks such as document-level translation and image generation.
- We design a local autoregressive translation (LAT) mechanism to enhance the decoding parallelizability. For each target decoding position, instead of only one token, we predict a short sequence of tokens in an autoregressive way and these short sequences at all translation positions are generated in parallel. Moreover, a simple but effective merging algo-

rithm is designed to merge these pieces into the final translation. This semi-autoregressive decoding mechanism improves the decoding parallelism and retains the high translation quality. This algorithm is easy to implement but it is worthwhile to tune the number of local translation steps to achieve the best accuracy and speed trade-off on specific datasets with certain efficiency requirements.

- We make the effort to close the gap between autoregressive translation (AT) models and fully non-autoregressive translation (NAT) models which generate sentences with just one single forward pass of neural networks. We first inspect the fundamental issue of fully NAT models, then adopt several various target dependency reduction techniques to build competitive fully NAT models which obtain new state-of-the-art results over other fully NAT models. Moreover, in contrast with AT systems, our proposed system obtains comparable performance and approximately $16.5\times$ speed-up at inference time. This model has the highest decoding speed with full parallelism among all models in this thesis while achieving competitive translation accuracy. We encourage researchers to try our models in different scenarios such as summarization and document-level translation. One concern is that if the sequence is very long such as documents, the dependency is more difficult to capture than sentence-level translation. At this time, the CTC beam-search (+ language model) is worth a try.
- We analyze the speed-accuracy trade-off of multilingual neural machine translation tasks through model capacity allocations, i.e., the deep encoder, shallow decoder (DESD) architecture. We find that for many-to-one translation we can indeed increase decoding speed without sacrificing quality using the DESD architecture, but for one-to-many translation, this structure causes a clear quality drop. To ameliorate this drop and retain high decoding speed, we propose a scheme of a deep encoder with multiple shallow decoders (DEMSD) where each shallow decoder is responsible for a disjoint subset of target languages. Experimental results show that our best DEMSD models achieve comparable translation accuracy and higher decoding speed. This method is intuitive and easy to implement and train. However, we can have a better trade-off with Mixture-of-Experts (MoEs).
- We investigate Mixture-of-Experts (MoEs) which can successfully increase the model capacity without a proportional increase in training computation. To meet the memory requirement at inference time, we introduce a language-level MoEs with the linguistic family constraint and successfully boost the performance of DESD on the one-to-many translation task while maintaining its high decoding speed. Furthermore, we revisit the capacity bottleneck of the shallow decoder and argue that the feed-forward network layer is key to mitigating this issue through an empirical study. We solely apply our proposed language-level MoEs to shallow decoders and show a superior performance and accuracy trade-off.

We argue that our method can generalize to multilingual tasks such as pre-training language models for language understanding and generation for the purpose of increasing capacity (better performance) and retaining the parameter efficiency at decoding time at the same time.

Experiments conducted in this thesis are based on the current hardware configuration such as CPU and GPU. In the future, there will be accelerators with larger memory and higher parallelism. Conclusions made in this thesis such as speed-ups may be different on them. However, we argue that the advantage of some proposed methods especially the non-autoregressive machine translation will become more distinct on more powerful hardware.

As deep-learning models are increasingly popular being parts of many real-world applications, it is important for these models to meet efficiency requirements of applications with strict latency requirements and limited memory resources. To achieve this goal, there are still open research questions which are worth further investigation along this line.

- First, efficiency matters especially in this era of larger models. Multilingual Language Models (MLLMs) such as mBERT [33], XLM-R [99] have emerged as a viable option for bringing the power of pretraining to a large number of languages. Several works try to apply pretrained MLLMs to machine translation tasks and find out surprisingly great performance, showing the strong ability of MLLMs to learn interlingual patterns to some extent. However, MLLMs refers to some largest models and running inference on these models is difficult. Several standard efficiency-related techniques such as pruning, knowledge distillation, factorization and quantization have been successfully applied to monolingual pretrained models without loss of accuracy on downstreaming tasks. We should call for more works to explore the trade-off between efficiency and accuracy of MLLMs. For instance, our language-level mixture of experts can be theoretically applied to many multilingual tasks to enhance both the training and decoding efficiency.
- Second, it is meaningful to extend proposed techniques to other tasks. In this thesis, we mainly apply proposed methods to machine translation tasks. However, we argue that these techniques can also be extended to other NLP and modality tasks such as automatic summarization, story generation, image and video generation. Recently, transformer has been applied to the image generation scenario and showed its strong ability to generate high-fidelity images such as ImageGPT [17] and DALL·E [145]. However, the generation process is inefficient due to its autoregressive property and a large number of pixels in images (very long sequences). Techniques mentioned in Part II can be worth a try and they will become more valuable on video generation which has much more generation steps.
- Third, decoding-efficient neural machine translation should embrace Simultaneous Neural Machine Translation. Simultaneous Neural Machine Translation is an essential application

for real-time understanding of conversations and lectures [43, 195]. This task requires that users receive translated sentences in an expeditious manner [121]. Several models have been built to solve this problem, which are mostly in the context of phrase-based machine translation. These methods receive the input incrementally, then decide when to send it to a MT system to translate the current segment independently [129], or with a minimal amount of language model context [9]. It is great to build Simultaneous Neural Machine Translation to achieve better the trade-off between translation accuracy and time delay.

Bibliography

[1] Roee Aharoni, Melvin Johnson, and Orhan Firat. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019. 2.1

[2] Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, et al. *Network flows: theory, algorithms, and applications*, volume 1. Prentice hall Englewood Cliffs, NJ, 1993. 3.3

[3] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, 2020. 4.5

[4] Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*, 2019. 2.1, 7.1, 7.4, 7.4, 8.4

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 4.2, 8.5

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 3.1, 5.1, 6.2, 7.1

[7] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. January 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015. 1, 2.1, 2.1

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015. 4.1

[9] Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour,

and Aura Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 437–445. Association for Computational Linguistics, 2012. 9

[10] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 4.1

[11] Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. 2.1

[12] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 8.2

[13] Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993. 1

[14] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign. In *Proc. of IWSLT*, 2014. 3.5

[15] William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. Kermit: Generative insertion-based modeling for sequences. *arXiv preprint arXiv:1906.01604*, 2019. 2.4, **??**

[16] William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. Imputer: Sequence modelling via imputation and dynamic programming. In *International Conference on Machine Learning*, pages 1403–1413. PMLR, 2020. 2.4

[17] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020. 9

[18] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016. 6.6

[19] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015. 3.5

[20] Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. Quora question pairs. *University of Waterloo*, 2018. 4.4

[21] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences

with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. 4.1

[22] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. 1, 5.1, 7.1

[23] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP 2014*, 2014. 2.1, 2.1

[24] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020. 4.1, 4.3, 4.5

[25] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations (ICLR)*, 2016. 4.3, 4.4

[26] Bernard Comrie. *Language universals and linguistic typology: Syntax and morphology*. University of Chicago press, 1989. 7.3

[27] Zihang Dai, Qizhe Xie, and Eduard Hovy. From credit assignment to entropy regularization: Two new algorithms for neural sequence prediction. *arXiv preprint arXiv:1804.10974*, 2018. 3.5, 3.5

[28] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019. 4.5

[29] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018. 4.5

[30] Yuntian Deng and Alexander Rush. Cascaded text generation with markov transformers. *Advances in Neural Information Processing Systems*, 33:170–181, 2020. 2.4, 18

[31] Yuntian Deng and Alexander Rush. Sequence-to-lattice models for fast translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3765–3772, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.318. URL `https://aclanthology.org/2021.findings-emnlp.318`. 2.4

[32] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380, 2014. (document), 3.5, 4.4, 5.3, 6.3, 6.4, 6.4

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. 2.1, 2.4, 4.1, 4.2, 4.4, **??**, 5.1, 5.2, 5.2, 5.2, 9

[34] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1166. URL `https://www.aclweb.org/anthology/P15-1166`. 7.1

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 4.1

[36] Matthew S. Dryer and Martin Haspelmath, editors. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2013. URL `https://wals.info/`. 7.3

[37] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013. 3.4

[38] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2019. 2.4

[39] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320*, 2020. 4.5

[40] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*. 8.1, 8.2

[41] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *NAACL*, 2016. 7.1

[42] Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-

Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144, 2011. 1

[43] Christian Fügen, Alex Waibel, and Muntsin Kolss. Simultaneous translation of lectures and speeches. *Machine Translation*, 21(4):209–252, 2007. 9

[44] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994. 2.2, 3.1, 3.3, 3.3

[45] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017. 2.1, 5.1, 7.1

[46] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China, November 2019. 2.4, 5.2, 5.5

[47] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019. (document), 5.1, 5.2, 5.2, 5.1, 5.3, 5.3

[48] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6114–6123, 2019. (document), 6.1, 6.2, 6.3, 6.3, **??**, 6.3, 6.4

[49] Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. Aligned cross entropy for non-autoregressive machine translation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3515–3523. PMLR, 2020. 2.4, 6.1, 6.2, 6.3, **??**

[50] Marjan Ghazvininejad, Omer Levy, and Luke Zettlemoyer. Semi-autoregressive training improves mask-predict decoding. *arXiv preprint arXiv:2001.08785*, 2020. **??**

[51] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for gpus. *arXiv preprint arXiv:1609.04309*, 2016. 3.4

[52] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connec-

tionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006. 6.3

[53] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017. 8.2

[54] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 4.5

[55] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, Canada, April 30-May 3, 2018, Conference Track Proceedings*, 2018. (document), 2.4, 5.1, 5.3, 6.1, 6.2, 6.2, 6.3, 6.3, 6.4, **??**, 6.5

[56] Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 11181–11191. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/675f9820626f5bc0afb47b57890b466e-Paper.pdf`. (document), 2.4, 6.1, 6.2, **??**, 6.3

[57] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012. 3.4

[58] Thanh-Le Ha, Jan Niehues, and Alexander Waibel. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*, 2016. 7.1

[59] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, et al. Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*, 2018. 1

[60] Hiroaki Hayashi, Yusuke Oda, Alexandra Birch, Ioannis Konstas, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Katsuhito Sudoh. Findings of the third workshop on neural generation and translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 1–14, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5601. URL `https://aclanthology.org/D19-5601`. 2.4

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 4.2

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc of CVPR*, pages 770–778, 2016. 3.5, 8.5

[63] Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W11-2123`. 6.4

[64] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2.4, 5.3, 6.3

[65] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc V Le. Transformer quality in linear time. *arXiv preprint arXiv:2202.10447*, 2022. 4.5

[66] RA Jacobs, MI Jordan, SJ Nowlan, and GE Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. 8.2

[67] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 7.3

[68] Melvin Johnson, Mike Schuster, Quoc Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017. 2.1, 7.1, 7.3

[69] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, et al. Marian: Fast neural machine translation in c++. *arXiv preprint arXiv:1804.00344*, 2018. 1, 2.4

[70] Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. In *International Conference on Machine Learning*, pages 2395–2404, 2018. 2.4, 6.3, **??**

[71] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proc. of CVPR*, pages 3128–3137, 2015. 3.5

[72] Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. Non-autoregressive machine translation with disentangled context transformer. In *International Conference on Machine Learning*, pages 5144–5155. PMLR, 2020. 6.2, **??**

[73] Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv*

preprint arXiv:2006.10369, 2020. (document), 1.1, 2.4, 6.1, 6.2, 6.4, 6.4, 6.3, 7, 7.1, 7.2, 8.1

[74] Zdeněk Kasner, Jindřich Libovickỳ, and Jindřich Helcl. Improving fluency of non-autoregressive machine translation. *arXiv preprint arXiv:2004.03227*, 2020. 6.4

[75] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020. 4.3

[76] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020. 4.1, 4.3, **??**, 4.4, 4.5

[77] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1139. URL `https://www.aclweb.org/anthology/D16-1139`. 5.3, 6.4

[78] Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, 2016. 2.4, 2.4, 6.3

[79] Yoon Kim, Yacine Jernite, David Sontag, and Alexander Rush. Character-aware neural language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 2.2

[80] Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, 2019. 2.4

[81] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3.5, 7.4, 8.4

[82] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`. 5.5

[83] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 4.4

[84] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings*

of the 2nd International Conference on Learning Representations (ICLR), number 2014, 2013. 6.3

[85] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019. 4.4

[86] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 4.1, 4.5

[87] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P17-4012`. 1

[88] Philipp Koehn. Statistical significance tests for machine translation evaluation. In *EMNLP*, pages 388–395, 2004. 2.1, 7.4, 7.4, 8.4

[89] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009. 1

[90] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, 2017. (document), 1, 2.3

[91] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003. 1, 4.4, 6.4

[92] Xian Kong, Adithya Renduchintala, James Cross, Yuqing Tang, Jiatao Gu, and Xian Li. Multilingual neural machine translation with deep encoder and multiple shallow decoders. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021. 8.1

[93] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. *Technical Report. University of Toronto*, 2009. 4.4

[94] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL `https://www.aclweb.org/anthology/D18-2012`. 6.4

[95] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent

subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018. 2.2

[96] Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3577–3599, 2021. 8.2

[97] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001. 18

[98] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, 2017. 4.4

[99] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019. 9

[100] Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. The significance of recall in automatic metrics for mt evaluation. In *Conference of the Association for Machine Translation in the Americas*, pages 134–143. Springer, 2004. 3.5, 2

[101] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378, 2017. 2.2

[102] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, 2018. 2.4, 5.3, 5.5, 6.2, 6.4, **??**

[103] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019. 4.3, 4.3, 4.5

[104] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020. 8.1, 8.2, 8.3, 8.4

[105] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966. 6.3

[106] M Paul Lewis. *Ethnologue: Languages of the world.* SIL international, 2009. 7.3

[107] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pages 6265–6274. PMLR, 2021. 8.2

[108] Xiang Li, Tao Qin, Jian Yang, Xiaolin Hu, and Tieyan Liu. Lightrnn: Memory and computation-efficient recurrent neural networks. In *NIPS*, pages 4385–4393, 2016. (document), 1.2, 3.3, 3.3, 3.1, 3.3, 3.3

[109] Zhuohan Li, Di He, Fei Tian, Tao Qin, Liwei Wang, and Tie-Yan Liu. Hint-based training for non-autoregressive translation. 2018. 6.2, **??**

[110] Jindřich Libovický and Jindřich Helcl. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1336. URL `https://www.aclweb.org/anthology/D18-1336`. 2.4, 6.2, 6.3, 6.4, **??**

[111] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014. 3.5

[112] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/ec8956637a99787bd197eacd77acce5e-Paper.pdf`. 4.4

[113] Hanxiao Liu, Zihang Dai, David So, and Quoc Le. Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34, 2021. 4.5

[114] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018. 4.1

[115] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 4.4, 4.4, **??**

[116] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015. 3.5

[117] Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. Flowseq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4273–4283, 2019. 2.4, 6.2, 6.3, **??**

[118] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011. 4.4

[119] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Possu Huang, and Richard Socher. Progen: Language modeling for protein generation. *bioRxiv*, 2020. 4.1

[120] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019. 2.4

[121] Takashi Mieno, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Speed or accuracy? a study in evaluation of simultaneous speech translation. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015. 9

[122] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 3.4

[123] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *NIPS*, pages 1081–1088, 2009. 3.4

[124] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012. 3.4

[125] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005. 3.4

[126] Makoto Nagao. A framework of a mechanical translation between japanese and english by analogy principle. 1

[127] Nikita Nangia and Samuel Bowman. Listops: A diagnostic dataset for latent tree learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 92–99, 2018. 4.3, 4.4

[128] Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, Xinyi Wang, and John Wieting. compare-mt: A tool for holistic comparison of language generation systems. *CoRR*, abs/1903.07926, 2019. URL `http://arxiv.org/abs/1903.07926`. 6.4

[129] Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Optimizing segmentation strategies for simultaneous speech translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 551–556, 2014. 9

[130] Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning*, 2018. 6.6

[131] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018. 4.1

[132] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019. 1, 4.4, 6.6

[133] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 3.5, 4.4, 5.3, 6.4, 7.2, 7.4, 8.4, 8.5

[134] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018. 4.1, 4.5

[135] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2020. 2.4, **??**

[136] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=QtTKTdVrFBB`. (document), 4.1, 4.3, 4.3, 4.3, 4.1, 4.4, 4.5

[137] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W18-6319`. 6.4

[138] Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. Glancing transformer for non-autoregressive neural machine translation. *arXiv preprint arXiv:2008.07905*, 2020. 2.4, 6.1, 6.3, 6.3, **??**, 6.6

[139] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie

Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019. 4.1

[140] Dragomir R Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47(4):919–944, 2013. 4.4

[141] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modeling. In *International Conference on Learning Representations (ICLR)*, 2020. 4.5

[142] Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*, 2020. 2.4

[143] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007. 4.3

[144] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016. 4.4

[145] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021. 9

[146] Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. Guiding non-autoregressive neural machine translation decoding with reordering information. *arXiv preprint arXiv:1911.02215*, 2019. **??**

[147] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. Comet: A neural framework for mt evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, 2020. 7.4, 8.4, 8.5

[148] Yi Ren, Jinglin Liu, Xu Tan, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. A study of non-autoregressive model for sequence generation. *arXiv preprint arXiv:2004.10454*, 2020. 6.2

[149] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. 4.1

[150] Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. Non-autoregressive machine translation with latent alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages

1098–1108, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.83. URL https://www.aclweb.org/anthology/2020.emnlp-main.83. 2.4, 6.1, 6.2, 6.3, **??**, **??**

[151] Víctor M. Sánchez-Cartagena, Marta Bañón, Sergio Ortiz-Rojas, and Gema Ramírez-Sánchez. Prompsit's submission to wmt 2018 parallel corpus filtering shared task. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, Brussels, Belgium, October . Association for Computational Linguistics. 6.4

[152] Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 291–301, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1029. URL https://www.aclweb.org/anthology/K16-1029. 2.4

[153] Jean Senellart, Dakun Zhang, Bo Wang, Guillaume Klein, Jean-Pierre Ramatchandirin, Josep Crego, and Alexander Rush. OpenNMT system description for WNMT 2018: 800 words/sec on a single-core CPU. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 122–128, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2715. URL https://www.aclweb.org/anthology/W18-2715. 1

[154] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162. 1.2, 2.2

[155] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proc. of ACL*, volume 1, pages 1715–1725, 2016. 3.1, 3.3, 3.3, 5.5

[156] Chenze Shao, Jinchao Zhang, Yang Feng, Fandong Meng, and Jie Zhou. Minimizing the bag-of-ngrams difference for non-autoregressive neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 198–205, 2020. 2.4, **??**

[157] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017. 3.4, 8.1, 8.2, 8.3, 8.4, 8.4

[158] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W

Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020. 4.5

[159] Xing Shi and Kevin Knight. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 574–579, 2017. 2.4

[160] Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. 2020. 2.4, 6.2, 6.3, **??**, 6.6

[161] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013. 4.4

[162] Harold Somers. Example-based machine translation. *Machine translation*, 14(2):113–157, 1999. 1

[163] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, pages 10107–10116, 2018. 2.4, **??**

[164] Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/stern19a.html`. 6.2, **??**

[165] Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. *arXiv preprint arXiv:1902.03249*, 2019. 2.4

[166] Zhiqing Sun and Yiming Yang. An em approach to non-autoregressive conditional sequence generation. In *International Conference on Machine Learning*, pages 9249–9258. PMLR, 2020. **??**

[167] Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhihong Deng. Fast structured decoding for sequence models. In *Advances in Neural Information Processing Systems*, pages 3011–3020, 2019. 2.4, 6.2, **??**

[168] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural

networks. In *NIPS*, pages 3104–3112, 2014. 1, 3.1, 5.1, 7.1

[169] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *NIPS 2014*, 2014. 2.1, 2.1, 2.1

[170] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, 2019. 4.4

[171] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR, 2020. 4.1

[172] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020. 4.1, 4.3, 4.5, 4.5

[173] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`. (document), 4.1, 4.1, 4.3, 4.4, 4.1, 4.4, 4.4, 4.7

[174] Jörg Tiedemann. Parallel data, tools and interfaces in opus. In *Lrec*, volume 2012, pages 2214–2218. Citeseer, 2012. 7.2

[175] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021. 4.5

[176] Peter Toma. Systran as a multilingual machine translation system. In *Proceedings of the Third European Congress on Information Systems and Networks, Overcoming the language barrier*, pages 569–581, 1977. 1

[177] Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. Engine: Energy-based inference networks for non-autoregressive machine translation. *arXiv preprint arXiv:2005.00850*, 2020. 2.4

[178] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017. 1, 1.1, 1.2, 2.1, 2.4, 3.1, 3.5, 4, 4.1, 4.2, 4.2, 4.2, 4.3, 4.4, 5.1, 5.3, 5.5, 6.2, 6.4, 6.6, 7.1, 7.2, 7.4, 8.4

[179] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 193–199, 2018. 4.2

[180] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proc. of CVPR*, pages 3156–3164. IEEE, 2015. 3.5

[181] Chengyi Wang, Shuangzhi Wu, and Shujie Liu. Accelerating transformer decoding via a hybrid of self-attention and recurrent neural network. *arXiv preprint arXiv:1909.02279*, 2019. 2.4

[182] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*, 2019. 2.1, 4.2, 4.3, 4.3, 4.5, 4.5, 7.1

[183] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 4.1, 4.3, 4.4, **??**

[184] Xinyi Wang, Yulia Tsvetkov, and Graham Neubig. Balancing training for multilingual neural machine translation. *arXiv preprint arXiv:2004.06748*, 2020. 7.2, 7.4, 7.4, 7.7

[185] Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5377–5384, 2019. 2.4, **??**

[186] Bingzhen Wei, Mingxuan Wang, Hao Zhou, Junyang Lin, and Xu Sun. Imitation learning for non-autoregressive neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1304–1312, 2019. 6.6

[187] David Weiss and Benjamin Taskar. Structured prediction cascades. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 916–923. JMLR Workshop and Conference Proceedings, 2010. 18

[188] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987. 6.4

[189] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019. 2.4

[190] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural

machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 1, 4.4

[191] Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. Tied transformers: Neural machine translation with shared encoder and decoder. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5466–5473, 2019. 4.3

[192] Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. Niutrans: an open source toolkit for phrase-based and syntax-based machine translation. In *Proceedings of the ACL 2012 System Demonstrations*, pages 19–24, 2012. 6.4

[193] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024*, 2019. 2.4

[194] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: a high-rank rnn language model. In *Proc. of ICLR*, 2018. 3.1, 3.2, 3.2, 3.4

[195] Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. Incremental segmentation and decoding strategies for simultaneous translation. In *IJCNLP*, pages 1032–1036, 2013. 9

[196] Weiqiu You, Simeng Sun, and Mohit Iyyer. Hard-coded gaussian attention for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7689–7700, 2020. 2.4

[197] Biao Zhang, Deyi Xiong, and Jinsong Su. Accelerating neural transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1789–1798, 2018. 2.4

[198] Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. *arXiv preprint arXiv:1908.11365*, 2019. 2.1, 7.1

[199] Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving massively multilingual neural machine translation and zero-shot translation. *arXiv preprint arXiv:2004.11867*, 2020. 2.1, 7.1, 7.4, 7.4, 8.4

[200] Jiacheng Zhang, Yanzhuo Ding, Shiqi Shen, Yong Cheng, Maosong Sun, Huanbo Luan, and Yang Liu. Thumt: An open source toolkit for neural machine translation. *arXiv preprint arXiv:1706.06415*, 2017. 3.5

[201] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2019. (document), 7.4, 7.2, 7.3, 8.4, 8.5

[202] Chunting Zhou, Jiatao Gu, and Graham Neubig. Understanding knowledge distillation in

non-autoregressive machine translation. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, Conference Track Proceedings*, 2020. 5.3, 6.3, 6.4, 6.4

[203] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015. 4.4